UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Expert-Coop
Um ambiente para desenvolvimento de Sistemas Multi-Agentes Cognitivos

Dissertação submetida à Universidade Federal de Santa Catarina
como requisito parcial à obtenção do grau de

Mestre em Engenharia Elétrica

por

Augusto Cesar Pinto Loureiro da Costa

Florianópolis, 06 de junho de 1997

Expert-Coop: Um ambiente para desenvolvimento de Sistemas Multi-Agentes Cognitivos

Augusto Cesar Pinto Loureiro da Costa

Esta dissertação foi julgada para a obtenção do título de **Mestre em Engenharia**, modalidade **Engenharia Elétrica**, área de concentração **Sistemas de Controle, Automação e Informática Industrial**, e aprovada em sua forma final pelo curso de Pós-Graduação.

Florianópolis, 06 de junho de 1997

Prof. Guilherme Bittencourt
Orientador

Prof. Adroaldo Raizer
Coordenador do curso de Pós-Graduação em Engenharia Elétrica

Banca Examinadora

Prof. Guilherme Bittencourt
Orientador

Prof. Carlos Alberto Maziero

Prof. Claudio Cesar de Sá

Prof. Marciano Morozowski Filho

A meu pai,
a Mônica, minha esposa.

Agradecimentos

Agradeço primeiramente ao prof. Guilherme Bitencourt pela oportunidade, pela confiança depositada, por todo o trabalho de orientacao e pela amizade cultivada durante este tempo.

Aos membros da banca examinadora, que contribuiram opinando e sugerindo.

Ao ainda ao prof. Edson de Piero, pela atenção dispensada durante a fase dos creditos.

Agradeço, especialmente ao prof. Carlos Alberto Maziero e a Eduardo Silva pelas sugestoes e pela atenção dispensada na solução dos problemas de infra-estrutura do Laboratorio.

A Ruben Cesar Macedo por toda atenção dispensada durante as idas a Curitiba, pelas sugestões bibliografica e pelas criticas.

A a Jaci Marta Pinto Loureiro da Costa, minha irmã, e Evilásio Teixeira Cardoso , meu sogro, que contribuiram com as correções gramaticais do texto final.

Aos funcionarios do Centro de Operações do Sistema da Eletrosul, Eng. José Ricardo Moreau, Eng. Assis Brasil Jr., Eng. Mauricio Costa, Jairo May e principalmente a Celio Jose Kopik que contribuiu com o conhecimento necessário para solucinar o problema da recomposição das linhas de transmissão de energia elétrica aqui tratado.

A especialmente a Edvaldo e Dair Machado que nos acolheu como a um membro de sua familia.

Agradeço aos amigos que fizeram parte desta jornada, em especial a Cristiane Paim, Evânio Nicoleit, Paulo James de Oliveira, Edvaldo Machado Jr., Max Mauro Dias, Lau Lung, Frank Siquera, Luis Gustavo Cavacanti, Lígia e Marcelo Correia, Jezer Pedrosa, Danielle Nishida, Emerson Raposo, Isabel Tonin, Cleto Leal, José Eduardo Góes, Conrado e Silene Seibel, Udo Fritzke e Carlos Montez.

Agradeço ainda a meu pai, meus irmãos e meus tios que mesmo tão distante sempre se fizeram tão perto. E principalmente a minha esposa que compartilhou comigo todas as emoções deste processo.

Resumo

Expert-Coop, um ambiente orientado a objeto, modular, concebido para auxiliar a implementação de Sistemas Multi-Agentes Cognitivos, abertos e heterogêneos, é apresentado neste trabalho. Construído a partir de um conjunto de bibliotecas de classes, que permitem a criação de agentes, segundo um modelo de arquitetura pré-definido. A arquitetura do agente, suportada pelo Expert-Coop, consiste em um modelo concorrente composto por seis processos que interagem utilizando troca de mensagens. São eles: mail-box, expert box, coordenador, sistema especialista (expert) e uma interface composta pelos processos terminal e "display". Os processos mail box e expert box implementam um mail box privado, também conhecido como "porto". O processo Coordinator è responsável pelo gerenciamento de todas as mensagens trocadas pelo processo (mensagens internas), bem como pelas mensagens trocadas pelos Agentes (mensagens externas). Cabe ainda ao Coordinator, o gerenciamento da cooperação entre os agentes da comunidade. O processo Expert provê a capacidade cognitiva do Agente. Este processo inclui um completo Arcabouço de Sistemas Especialistas chamado FASE. O processo Interface, que consiste em um display para visualização das mensagens e um terminal para envio destas, propicia a comunicação entre o Agente e o usuário. Entretanto é possível a utilização de módulos gráficos baseados nas bibliotecas do Motif customizadas à aplicação em questão. O ambiente foi codificado em LISP/ CLOS, e sua modularidade, assegurada pela metodologia de Orientação a Objetos utilizada, permite facilmente adicionar novas características ao ambiente. O Expert-Coop vem sendo utilizado para implementação de um Sistema Multi-Agentes destinado a auxiliar o processo de recomposição da rede de transmissão de energia elétrica da região do sul do Brasil, após a ocorrência de um "blackout".

Abstract

A modular object-oriented environment, called Expert-Coop, designed to help the implementation of open heterogeneous cognitive Multi-Agent Systems is presented in this work. The environment consists in a library of classes that supports the creation of agents according to a predefined architecture. This architecture consists of the following processes: mail box, expert box, coordinator, expert system and interface. The mail box and expert box processes just receive and stores input messages. The coordinator process reads the input messages and controls all message exchange between internal processes or with other agents. The expert system process provides the cognitive capabilities of the agent. The environment includes a complete expert system shell, called FASE. The interface process allows the communication between the agent and an external user. The agent interface is fully customizable, any Motif based interface can be easily integrated into the environment. By default, the interface consists of a terminal and two output displays. Through the terminal the user may interact with the local expert system or propose a task to be solved by the agent community. The output displays show, respectively, the recently exchanged messages and the status of the agent's current tasks. The environment is written in Common Lisp/CLOS and its modularity, guaranteed by the object-oriented design methodology, makes it easy to modify and augment it with new features. The environment is being used to develop an application concerning the recomposition of an electricity transmission network, covering Brazil's south region, after a blackout has affected part of the network.

Índice

Introdução

"O conhecimento pelo Conhecimento" - eis a última armadilha colocada pela moral: é assim que mais uma vez nos enredamos inteiramente nela.
Friedrich Nietzsche
Além do Bem e do Mal.

Os computadores digitais surgiram por volta da década de quarenta, como sendo máquinas capazes de manipular números e símbolos, como os seres humanos, entretanto com maior rapidez e confiabilidade. A essas máquinas foram adicionados os chamados programas computacionais que permitiam a combinação dessa capacidade de manipular números e símbolos, processar dados, e realizar uma diversidade de tarefas de caráter procedural , que se caracterizavam pelo processamento exaustivo.

A Inteligência Artificial (IA), surgiu em meados da década de cinquenta, com o propósito de dotar os computadores digitais da capacidade de representar e manipular conhecimento, além do simples armazenamento e manipulação de dados. Essa nova característica permitiria simular a inteligência humana nos computadores digitais, tornando-os capazes de solucionar problemas que requeriam um comportamento não mais procedural e sim inteligente, tal como jogar xadrez com a habilidade de um enxadrista. O desconhecimento dos princípios que fundamentavam a inteligência e dos limites práticos da capacidade de processamento dos computadores levaram a promessas exageradas e às correspondentes decepções.

A Inteligência Artificial não possui uma definição formal precisa, mesmo porque isso implicaria uma definição formal para inteligência. Entretanto, foram propostas algumas definições operacionais para a Inteligência Artificial: "uma máquina é inteligente se ela é capaz de solucionar uma classe de problemas que requerem inteligência para serem solucionados por seres humanos" [MH69]; "Inteligência Artificial é a parte da ciência da computação que compreende o projeto de sistemas computacionais que exibam características associadas, quando presentes no comportamento humano, à inteligência" [BF81]; ou ainda "Inteligência Artificial é o estudo das faculdades mentais através do uso

de modelos computacionais".

Os objetivos centrais da IA podem ser sintetizados a nível teórico como a criação de teorias e modelos para a capacidade cognitiva e a nível prático como sendo a implementação de sistemas computacionais baseados em tais modelos. Neste sentido, a IA tem uma relação com seus objetos de estudo semelhante a da psicologia, mas com uma importante diferença: os modelos e teorias da IA são implementados em um computador, atribuindo a estes uma certa autonomia. Assim, a validade de um modelo ou de uma teoria de IA dispensa a prova comparativa de seus resultados com o comportamento psíquico humano, como no caso da psicologia, podendo assim ser implementada em um computador e demonstrada através da ação inteligente do programa no mundo [Bit96].

Existem basicamente duas linhas de pesquisa destinadas à construção de sistemas inteligentes: *Conexionista e Simbólica.* A linha *Conexionista* visa a modelagem da inteligência humana através da simulação dos componentes do cérebro, os neurônios e suas interligações também conhecidas como as redes neurais ou neuronais. A linha *Simbólica* baseia-se na lógica e teve McCarthy e Newell como seus principais defensores. Os princípios desta linha de pesquisa foram apresentados no artigo "Physical Symbol System" de Newell [New80].

Os Sistemas Especialistas, programas computacionais concebidos para simular a perícia de um especialista, ainda que em um domínio restrito, alcançaram enorme sucesso na década de 70 e consolidaram a manipulação simbólica de um grande número de fatos especializados sobre um determinado domínio, como sendo o paradigma corrente para a construção de sistemas inteligentes do tipo simbólico. Esses sistemas apresentavam, entretanto, uma concepção centralizada do conhecimento, caracterizando assim um *Comportamento Individual* em relação ao ambiente.

No final dos anos 70 esse perfil centralizador foi contraposto, por um lado pelos trabalhos da linha conexionista utilizando redes neurais. Estes trabalhos apresentavam distribuição e paralelismo, e paralelamente os sistemas simbólicos baseados no chamado *Modelo de Quadro Negro* (do inglês "Blackboard Model") [HR85], caracterizados por um controle distribuído. O aparecimento desses trabalhos apresentando uma proposta de distribuição em IA, aliado à maturidade alcançada pelas tecnologias das Redes de Computadores e dos Sistemas Distribuídos deram origem a Inteligência Artificial Distribuída, *(IAD).*

A *IAD* representa hoje uma das áreas da IA tradicional que apresenta um grande potencial em aplicações do mundo real [Dur91]. Esta área dedica-se ao estudo de teorias e implicações práticas a respeito de como reunir um determinado conjunto de Agentes Computacionais em uma Comunidade para solucionar problemas eminentemente distribuídos ou utilizar a robustez das redes de computadores para tratamento de problemas complexos, como por exemplo: controle de tráfego aéreo, monitoramento ambiental, gerenciamento de rede de computadores, distribuição de recursos como água, energia elétrica, etc. Devido a motivos históricos a IAD dividiu-se em dois enfoques: a Solução

Distribuída de Problemas (SDP) e os Sistemas Multi-Agentes (SMA). A SDP concentrou seus esforços no problema a ser solucionado e em como utilizar os recursos disponíveis nas redes de computadores para tal fim. Os SMA por outro lado concentram-se no Agente e em suas propriedades, principalmente aquelas que o levam a cooperar [CL87] [Sic96].

O problema que originalmente motivou esse trabalho de pesquisa em SMA foi o processo de recomposição da redes de transmissão de energia elétrica da região sul do Brasil, após esta rede, ou parte dela, ter sido afetada por um "blackout". No problema em questão, cada uma das unidades da rede de transmissão de energia elétrica - usina hidroelétrica, termoelétrica ou subestação - possui seu próprio domínio de conhecimento sobre os equipamentos e procedimentos operacionais das respectivas unidades (ex. grupo-gerador, linha de transmissão, disjuntores, barramentos, etc). Entretanto, informações globais tais como a completa topologia da rede não pertencem a esse domínio de conhecimento. O processo de recomposição da rede de transmissão de energia é realizado atualmente pelos operadores das respectivas unidades, que seguem um conjunto de procedimentos pré-determinados de acordo com o estado específico da unidade na rede. Tais procedimentos podem ser codificados em um sistema especialista. Esses procedimentos locais são coordenados por uma unidade central capaz de monitorar toda a rede. Por razões práticas esses procedimentos visam minimizar a comunicação entre os operadores das unidades em detrimento de uma operacionalidade ótima do sistema. Esse problema da recomposição da malha de transmissão de energia elétrica, caracteriza uma das situações do mundo real em que agentes inteligentes com seus respectivos domínios de conhecimento, necessitam cooperar para atingir uma meta global.

Um ambiente para desenvolvimento de Sistemas Multi-Agentes Cognitivo, chamado Expert-Coop, e os resultados de sua aplicação para solucionar alguns casos pertencentes ao problema da recomposição da malha de transmissão de energia elétrica da região sul do Brasil, são apresentados nessa dissertação. O ambiente suporta Sistemas Multi-Agentes Cognitivos, Heterogêneos e Abertos projetados para cooperar, mesmo que os agentes pertencentes à comunidade não possuam uma representação do conhecimento global desta. Nessa comunidade de agentes, as metas globais a serem atingidas, através da cooperação de seus membros, surgem a partir das necessidades particulares de cada um do agentes. Quando um determinado agente necessita realizar uma tarefa, este abre uma licitação e promove a difusão do edital da licitação na comunidade. A licitação é mantida aberta até que cada um dos agentes ativos deposite a sua proposta. Uma vez depositadas todas as propostas, um vencedor é definido, sendo informado de que ganhou a licitação, assumindo assim a responsabilidade pela tarefa licitada, e os demais são informados do fechamento da licitação. As tarefas licitadas podem ser realizadas por diferentes agentes, não necessariamente todos, com diferentes esforços computacionais. Os agentes pertencentes a essa comunidade podem entrar e sair a qualquer instante, entretanto, para deixar a comunidade o agente precisa encontrar um outro agente ou outros agentes que possam assumir a realização das suas tarefas. Nessa comunidade de agentes, todos os seus participantes possuem o mesmo nível hierárquico e podem abrir processos licitatórios concorrentemente. Esse ambiente foi codificado utilizando *CMU Common Lisp* [MaC92], uma implementação de domínio publico do *Common Lisp*

[SJ84] realizado pela Carnegie-Mellon University, que inclui o PCL, uma implementação do *CLOS (Common Lisp Object System)* (extensão Orientada a Objetos do Common Lisp) e um "package" específico, chamado "wire", que permite a comunicação entre processos LISP utilizando os SOCKETS do UNIX no domínio INET. O Expert-Coop está disponível através de ftp anônimo e é compatível com os seguintes sistemas operacionais: SunOS, Solaris, HP-UX, Irix, FreeBSD e Linux.

A dissertação está dividida em seis capítulos. No primeiro capítulo é feita uma abordagem sobre os Sistemas Especialistas e o FASE, uma Ferramenta para construção de Arcabouços de Sistemas Especialistas, que serviu como ponto de partida para esse trabalho e atualmente é responsável pelo carater cognitivo do agente, é apresentada. No capítulo 2 é feita uma apresentação da Inteligência Artificial Distribuída, começando pela sua origem e abordando as linhas de pesquisa desenvolvidas atualmente: a Solução Distribuída de Problemas e os Sistemas Multi-Agentes. Os Sistemas Multi-Agentes são vistos no capítulo 3 com uma maior profundidade. O capítulo 4 apresenta o Expert-Coop abordando a Arquitetura do Agente Cognitivo suportado pelo Ambiente, a estratégia de cooperação utilizada, bem como a linguagem para comunicação dos agentes utilizada no ambiente. Uma implementação de um Sistema Multi-Agentes Cognitivo, utilizando o Expert-Coop, e destinado a solucionar alguns casos do problema da recomposição de parte da malha de transmissão de energia elétrica da região sul, é apresentada no capítulo 5. Finalmente, as conclusões e perspectivas de futuros trabalhos são abordados no capítulo 6.

Capítulo 1

Sistemas Especialistas

1.1 Introdução

Neste capítulo são apresentados os Sistemas Especialistas (SE), programas de computador que têm como objetivo simular a perícia de especialistas humanos, solucionando problemas do mundo real, em um domínio restrito. A apresentação inicia com o formalismo dos Sistemas de Produção, considerados os precursores dos *SE*. Em seguida, a questão da aquisição de conhecimentos é abordada, apresentando-se os principais métodos de representação de conhecimento, ilustrados através de exemplos do sistema FASE [Bit94]. Apresentam-se ainda os principais conceitos envolvidos na utilização de ferramentas para construção de SE e finalmente uma seção é dedicada a uma breve apresentação do FASE, uma ferramenta para construção de sistemas especialista, baseada no LISP, de domínio público, disponivel via ftp anônimo em *ftp.lcmi.ufsc.br /pub/diversos/ia/fase.tar.gz* .

1.2 Sistemas de Produção

Os Sistemas de Produção caracterizam-se pela utilização das chamadas *Regras de Produção*, expressões que associam a cada condição uma ação. A utilização de pares de expressões consistindo em uma condição e uma ação foi introduzida por Post em 1936, quando propôs o que atualmente é conhecido por *Sistemas de Post* [Pos43]. Um sistema de Post consiste em um conjunto de regras para especificação sintática de transformações sobre uma cadeia de caracteres, representando assim um método geral para processamento de dados.

Um Sistema de Produção, em sua forma mais simples, apresenta dois componentes passivos, *Conjunto de Regras* e *Memória de Trabalho*. Estes são definidos da seguinte forma:

- Regras de Produção: conjunto ordenado de pares (LHS, RHS) onde LHS e RHS são seqüência de caracteres.

- Memória de Trabalho: uma seqüência de caracteres.

Os *Sistemas de Produção* apresentam ainda, um componente ativo, *Interpretador*, responsável pela realização do seguinte procedimento:

- Interpretador: Para cada regra (LHS, RHS), se a seqüência de caracteres LHS está contida na memória de trabalho, então os caracteres LHS são substituídos na memória de trabalho pelos caracteres de RHS, senão continua na próxima regra.

Exemplo: Seja o *Sistema de Produção* especificado segundo o seguinte conjunto de regras:

$$\mathrm{S} \longrightarrow ABA, A \longrightarrow A1, A \longrightarrow 1, B \longrightarrow B0, B \longrightarrow 0$$

Dado que o caractere S está armazenado na memória de trabalho, a execução do procedimento interpretador definido acima resultaria na seguinte seqüência de valores como conteúdo da memória de trabalho:

$$\mathrm{S} \xrightarrow{1} ABA \xrightarrow{2} A1BA \xrightarrow{3} 11BA \xrightarrow{4} 11B0A \xrightarrow{5} 1100AA \xrightarrow{6} 1100A1 \xrightarrow{7} 110011$$

Os sistemas de produção foram redescobertos durante os anos setenta como uma ferramenta para a modelagem da psicologia humana. O formato condição-ação adapta-se à modelagem de todos os comportamentos baseados em pares estímulo-resposta. Dois sistemas que utilizaram o modelo de sistemas de produção para a modelagem do comportamento humano foram PASII [Wat73] e VIS [Mor73].

1.3 Sistemas Especialistas

Os Sistemas Especialistas, são programas de computador, concebidos para reproduzir o comportamento de especialistas humanos na solução de problemas do mundo real, mas o domínio desses problemas é altamente restrito. Os primeiros *SE* a alcançar sucesso foram o sistema DENDRAL [FBL71] e MYCIN [Sho76], ambos na década de setenta. O sistema DENDRAL foi concebido para inferir a estrutura molecular de compostos desconhecidos, a partir de dados espectrais de massa e de resposta magnética nuclear. O MYCIN, por sua vez, foi implementado para auxiliar médicos na escolha de uma terapia de antibióticos para pacientes com bacteremia, meningite e cistite infecciosa, em ambiente hospitalar.

Desde então, muitos *SE* foram desenvolvidos para solucionar problemas em diversos domínios, dentre eles: agricultura, química, sistemas de computação, eletrônica, engenharia, geologia, gerenciamento de informação, direito, matemática, medicina, aplicações militares, física, controle de processos e tecnologia aeroespacial.

1.3.1 Arquitetura de Sistemas Especialistas

A arquitetura simples dos *Sistemas de Produção* de Post foi generalizada, adquirindo uma maior eficiência e expressividade, dando origem a Arquitetura dos Sistemas Especialistas, como mostra a figura 1.1. Atualmente um *SE* apresenta, em geral, uma arquitetura composta por três módulos: *Base de Regras, Memória de Trabalho ou Base de Fatos e Motor de Inferência.* A Base de Regras mais a Memória de Trabalho foram chamadas de *Base de Conhecimento* do *SE*, onde se encontra representado o conhecimento sobre o domínio em questão. O Motor de Inferência é o mecanismo de controle do sistema, responsável pela avaliação e aplicação das regras segundo as informações contidas na Memória de Trabalho.

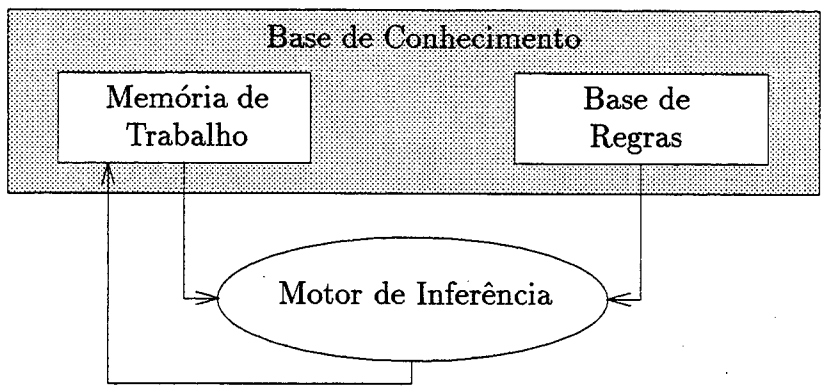

Figura 1.1: Arquitetura de um Sistema Especialaista

A Memória de Trabalho, que no modelo de Post continha apenas uma seqüência de caracteres, pode conter qualquer tipo de estrutura de dados. Mais do que estrutura de dados, a informação contida na memória de trabalho de um *SE* deve respeitar um *Método de Representação de Conhecimento*, ou seja, uma linguagem formal e uma descrição matemática de seu significado. A lógica de primeira ordem é um exemplo típico de representação de conhecimento.

A Base de Regras passa a conter condições que representam "perguntas" à representação de conhecimento armazenada na Memória de Trabalho. Tais perguntas, que se limitavam a uma simples comparação de carácteres no modelo de Post, podem ser de diferentes tipos, mas geralmente envolvem variáveis a serem instanciadas e eventualmente algum tipo de inferência. A sintaxe das regras é definida pelo sistema em questão podendo até ser bastante flexível, aproximando-se da linguagem natural, como no caso dos sistemas OPS [FHRSW82] e G2 [Gen92], ou bastante formal, como na família de sistemas OPS [FM77]. Outra característica comum nos *SE* atuais é a existência de um mecanismo de raciocínio incerto a respeito do conhecimento do domínio.

O Motor de Inferência controla a atividade do sistema. Esta atividade ocorre em ciclos compostos por três fases:

- Correspondência de dados, onde são selecionadas as regras que satisfazem a situação atual.
- Resolução de Conflitos, onde são escolhidas e ordenadas as regras que serão realmente utilizadas dentre as que foram selecionadas na primeira fase.
- Ação, a execução propriamente dita das regras.

As principais vantagens dos sistemas de produção são [Wat86]: modularidade, uniformidade, naturalidade. As principais desvantagens são: ineficiência em tempo de execução, e complexidade do fluxo de controle que conduz a solução do problema. Estas vantagens e desvantagens determinam as características requeridas dos domínios que se adaptam ao desenvolvimento de *SE's* baseados em sistemas de produção: ser descrito por conhecimento consistindo por um conjunto muito grande de fatos parcialmente independentes, dispor de métodos de soluções compostos por ações independentes e apresentar uma nítida separação entre conhecimento e ação.

O bom desempenho de um *SE* está intimamente ligado ao conhecimento armazenado nas suas regras e na memória de trabalho. Este conhecimento deve ser obtido junto a um especialista humano do domínio e representado de acordo com regras formais definidas para a codificação da base de regras do *SE's* em questão. Isto divide um *SE* em duas partes: a ferramenta de programação que define o formato do conhecimento na base de conhecimento, memória de trabalho e base de regras, além dos aspectos operacionais de sua utilização, e o conhecimento do domínio propriamente dito.

Devido a essa separação, atualmente os *SE* são desenvolvidos em geral a partir de *Arcabouços de Sistemas Especialistas ASE*: ferramentas que suportam todas as funcionalidades de um *SE* [Gev87], restando ao programador apenas codificar o conhecimento especializado de acordo com a linguagem de representação de conhecimento disponível. A existência de *ASE* facilitou bastante a implementação de *SE* e foi um dos fatores responsáveis por sua disseminação.

1.4 Aquisição de Conhecimento

A parte mais sensível no desenvolvimento de um *SE* é certamente a *Aquisição de Conhecimento*. Mais do que limitar-se à simples adição de novos elementos à Base de Conhecimento, faz-se necessária a integração do novo conhecimento ao já existente. Essa integração consiste em definir as relações entre os elementos que constituem o novo conhecimento e o já armazenado na base. Dois tipos de mecanismo para a definição de tais relações foram propostos: ligar os elementos de conhecimento diretamente através

de ponteiros e reunir diversos elementos relacionados em grupos.

Outro ponto importante na aquisição de conhecimento é o tratamento de incoerências. O processo de aquisição de conhecimento pode levar a aquisição de erros, quer sejam eles inerentes ao próprio conhecimento adquirido, como no caso de dados obtidos através de sensores sujeitos a ruídos, ou podem ainda ser gerados pela interface humana existente entre o mundo real e o sistema de representação. Algumas técnicas foram definidas justamente para evitar tais erros de aquisição, como por exemplo, a especificação de regras de aquisição que definam o tipo de conhecimento esperado. Tais técnicas são comuns aos sistemas de representação de conhecimento e aos sistemas de gerenciamento de bancos de dados. Por um outro lado, uma base de conhecimento pode ser examinada periodicamente com a finalidade de detectar incoerências eventualmente introduzidas no processo de aquisição. Este método é limitado devido ao fato de as linguagens de representação razoavelmente expressivas não disporem de procedimentos completos de verificação conhecidos. Finalmente, deve-se salientar que a adequação do formalismo de representação de conhecimento ao tipo de conhecimento do mundo real a ser representado é fundamental para a eficiência do processo de aquisição.

1.5 Métodos para Representação do Conhecimento

A representação do conhecimento corresponde à parte mais importante em um projeto de *Sistemas Especialistas*. A escolha do método a ser utilizado para representar o conhecimento do domínio em questão irá influenciar diretamente no desempenho do *SE*. A linguagem associada ao método escolhido para a representação do conhecimento em questão, deve ser suficientemente expressiva para que tal representação seja realizada de forma completa e eficiente. Em tese, uma representação geral como a lógica seria suficientemente expressiva para representar qualquer tipo de conhecimento. Entretanto, fatores como eficiência, facilidade de uso e a necessidade de expressar conhecimento incerto e incompleto, levaram ao desenvolvimento de diversos formalismos para representação de conhecimento. A seguir são apresentados alguns dos formalismos de representação de conhecimento mais utilizados. Para cada um dos formalismos apresentados é mostrado um exemplo ilustrativo, utilizando-se a sintaxe do *FASE*.

1.5.1 Lógica

A lógica é um formalismo largamente utilizado em sistemas especialistas. As duas principais vantagens deste tipo de formalismo são: a expressividade inerente da linguagem e uma semântica bem definida e bastante estudada. Uma base de conhecimento construída utilizando-se a lógica como formalismo, consiste em um conjunto de fórmulas lógicas, que podem ser representadas, por exemplo, em expressões LISP. Este formalismo adequa-se a domínios onde o conhecimento é largamente desestruturado e consiste em uma coleção de fatos independentes. O método de inferência pode ser uma simples

unificação ou dedução automática.

Exemplo: Deseja-se representar as condições de funcionamento de um motor de automóvel a partir dos seguintes indicadores:

Alto consumo de combustível.
Baixo rendimento do motor.
Baixo consumo de óleo do cárter.
Viscosidade do óleo do cárter moderada.
Não apresenta centelhamento no escape.
Não apresenta ruído no motor.
A taxa de compressão do motor é moderada.
Não apresenta odor de combustível no escapamento.
Tomada de aceleração é normal.
Partida rápida.

A informação desejada assume a seguinte sintaxe, no *FASE*, quando é utilizada lógica como linguagem de representação de conhecimento:

```
((logic  (consumo-de-combustivel alto)
         (rendimento-do-motor baixo)
         (consumo-do-oleo-do-carter baixo)
         (viscosidade-do-oleo-carter moderada)
         (centelha-no-escape negativo)
         (ruido-no-motor negativo)
         (taxa-de-compressao-do-motor moderada)
         (odor-de-combustivel-no-escape negativo)
         (tomada-de-aceleracao normal)
         (partida-do-motor rapida)))
```

1.5.2 Quadros

Os Quadros (do inglês *"frames"*) foram propostos por Marvin Minsky em 1975,, como um formalismo de representação de conhecimento. Uma base de conhecimento construída utilizando-se este formalismo consiste de uma hierarquia de estruturas de dados chamada de quadros. Cada quadro possui um conjunto de atributos onde um valor pode ser associado a cada atributo. Este valor pode ser qualquer informação primitiva sobre o conceito representado pelo quadro, ou um ponteiro para um outro quadro. Existem três mecanismos de inferência integrados no formalismo: herança de valores de atributos através da hierarquia, valores previamente escolhidos ("default") podem ser utilizados quando não há informação disponível e ligação procedural para permitir a execução de uma função externa ao formalismo. Este tipo de formalismo é adequado para domínios estruturados taxonomicamente, onde o mecanismo de herança pode ser explorado eficientemente.

Exemplo: Deseja se representar o conhecimento a respeito da Subestação de Transmissão de Energia Elétrica Ivaiporã, utilizando-se quadros como formalismo para representação de conhecimento. A Subestação possui os seguintes equipamentos:

- Dois barramentos A e B.
- Oito disjuntores: dj1010, dj1012, dj1022, dj1030, dj1032, dj1082, dj1090, dj1092.
- Cinco Linhas de Transmissão: uhss, are, lon, stid-1, stid-2.

Para os barramentos e disjuntores é suficinte representar os estados em que cada um deles se encontra, ligado ou desligado. Entretanto, para as linhas de transmissão é necessario representar um conhecimento mais abrangente, informando: a tensão, a freqüência, e o ângulo de fase. Outra informação importante a ser representada corresponde aos valores da tensão em módulo e ângulo, freqüência, nominais e instantâneos da própria subestação.

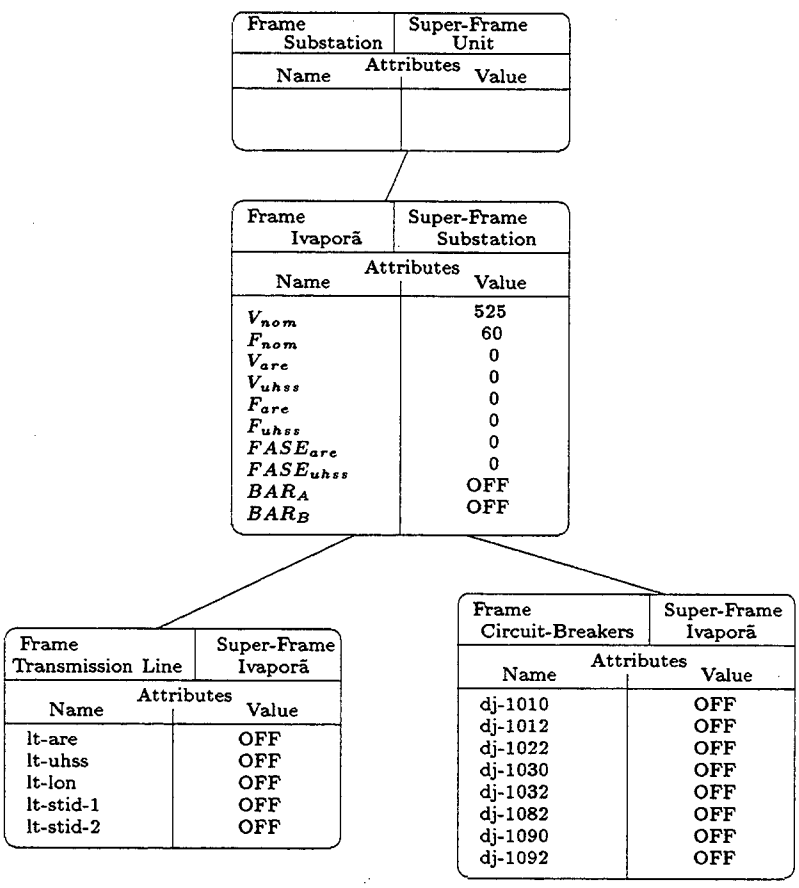

Figura 1.2: A Subestação Ivaiporã, segundo o formalismo dos quadros.

Esse conhecimento sobre a subestação de Ivaiporã, representado da figura 1.2, apresenta a seguinte sintaxe quando representado no *FASE*.

```
((frame (subestação root)
          (ivaporan subestação
            (v-nom 525)(fnom 60)(f-uhss 0)(f-are 0)(v-are 0)
            (v-uhss 0)(fase-are 0)(fase-uhss 0)(localizacao area-1)
            (barramento-A desligado)(barramento-B desligado))

           (LT Ivaporan  (lt-uhss desligado)(lt-are desligado)
                         (lt-lon desligado)(lt-stid-1 desligado)
                         (ltstid-2 desligado))

           (DJ Ivaporan  (dj1010 desligado)(dj1090 desligado)
                         (dj1030 desligado)(dj1032 desligado)
                         (dj1012 desligado)(dj1092 desligado)
                         (dj1082 desligado)(dj1022 desligado))))
```

A linguagem de quadros acima permite a definição de estruturas complexas de dados com procedimentos complexos de herança através de hierarquia e facilidades de ligações procedurais.

1.5.3 Redes Semânticas

As redes semânticas foram introduzidas por Quillian em 1968, como um modelo para a memória associativa do ser humano. Este formalismo consiste em um grafo orientado. Os nodos podem ser utilizados para representar conceitos primitivos, predicados, objetos, etc. Os arcos são utilizados para associar os nodos com relações binárias. Os arcos podem ser positivos ou negativos permitindo assim a representação explícita de exceções.

O mecanismo de inferência básico é a herança através dos arcos do grafo. Na ausência de arcos negativos o mecanismo é simples e eficiente. A introdução de arcos negativos, representando exceções, exige um maior cuidado para que os resultados do mecanismo de inferência correspondam à interpretação intuitiva desejada.

Exemplo: Seja a seguinte informação a ser representada utilizando-se o formalismo das redes semânticas: Clyde é um elefante real, a cor dos elefantes é cinza. Entretanto os elefantes reais não são cinza. A figura 1.3, apresenta uma rede semântica contendo a informação envolvida no problema "A cor de Clyde".

A sintaxe desta Informação, utilizando as Redes Semânticas é do *FASE* apresentada abaixo.

```
(snet (cor (clyde) (real) (elefante) (cinza)
           (clyde real e-um )
           (real elefante e-um)
           (elefante cinza e-um)
           (real cinza (not . e-um))))
```

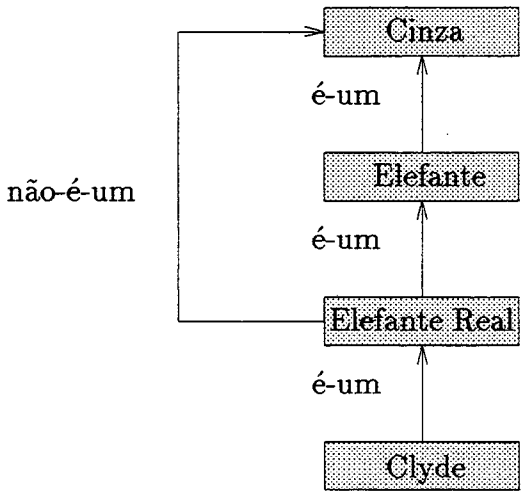

Figura 1.3: Rede semântica a "A cor de Clyde"

A principal utilização do formalismo de redes semânticas é para introduzir exceções em redes de herança.

1.6 FASE, Ferramenta para Arcabouços de Sistemas Especialistas

O *FASE* [BM93] é uma ferramenta que permite construir arcabouços de sistemas especialistas adaptado às características do problema a ser resolvido. O ambiente é modular, constituído por uma biblioteca de funções LISP, dividida em quatro módulos : Representação de Conhecimento, Estratégia de Controle, Resolução de Conflitos e Tratamento de Incertezas. Existe ainda um módulo único para manipulação de bases de regras ao qual podem ser integrados os recursos disponíveis nos módulos da biblioteca de funções. A integração desses recursos ao módulo manipulador de bases de regras resulta no arcabouço para sistemas especialistas desejado. A arquitetura do *FASE* é apresentada na figura 1.4.

1.6.1 Como se utiliza o FASE

A utilização do FASE consiste basicamente de duas etapas: na primeira ocorre a construção do núcleo, na segunda, constrói-se o sistema especialista para solucionar o problema desejado.

A construção do núcleo é a etapa mais importante, pois ali serão determinadas as características do motor de Inferência a ser utilizado no Sistema Especialista. Esta construção se dá selecionando as bibliotecas de função adequadas ao problema em questão e na compilação destas resultando no Motor de Inferência a ser utilizado na segunda etapa, a construção do sistema especialista propriamente dito.

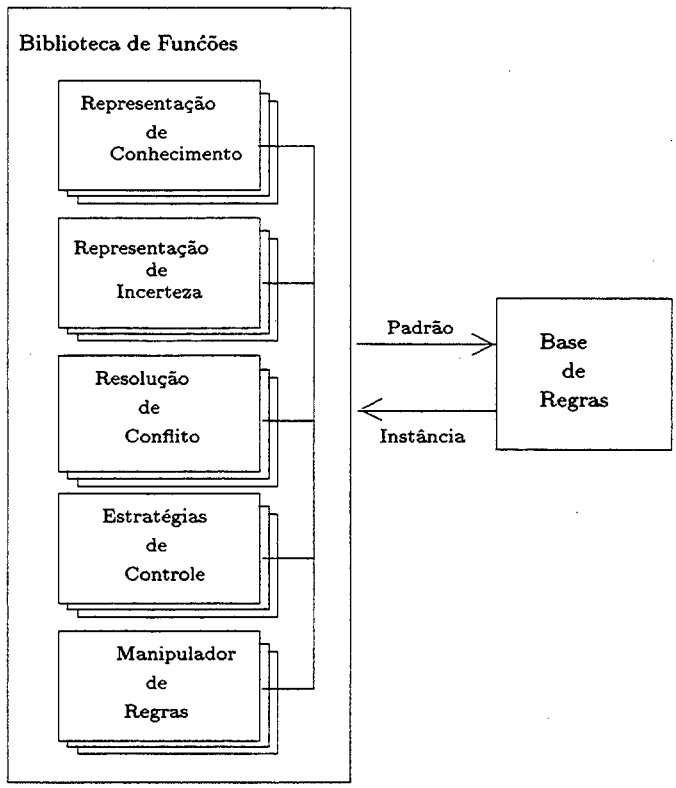

Figura 1.4: Arquitetura do FASE

A integração dos diferentes módulos na construção de um arcabouço de sistema especialista coerente é atingida, uniformizando-se uma interface para os mesmos tipos de módulos e definindo-se uma interface adequada para os diferentes tipos de módulos.

1.6.2 Biblioteca de Funções do FASE

Manipulador de Regras

Existe apenas um módulo de manipulação de base de regras. O formalismo implementado neste módulo define as interfaces entre ele e os demais módulos. Cada regra consiste em um par de listas de padrões, o lado direito da regra e o lado esquerdo da regra , onde cada padrão é uma expressão LISP, normalmente contendo variáveis. O resultado da interação de um padrão com a base de conhecimento é uma lista de substituições, que corresponde às instâncias dos padrões presentes na base de conhecimento. Tais substituições são aplicadas no lado oposto da regra. A interpretação deste padrões depende do formalismo utilizado para representar o conhecimento e da estratégia de controle utilizada.

Resolução de Conflitos

O módulo de resolução de conflitos permite ao usuário definir a maneira como as regras que satisfazem a situação atual são selecionadas para execução. O FASE dispõe das seguintes estratégias:

- ordem lexicográfica sobre os nomes das regras,
- regra mais recente,
- regra mais específica.

Estas estratégias podem ser usadas isoladas ou combinadas, desde que uma relação de precedência sobre as estratégias seja estabelecida.

Estratégias de Controle.

Encontram-se disponíveis no FASE dois tipos de estratégias de controle para encadeamento de regras:

- **Progressiva** (para frente).Na estratégia progressiva, para frente, formula-se o problema sob a forma de uma situação inicial, armazenando-a na memória de trabalho. Os padrões esquerdos das regras são comparados com o conteúdo da memória de trabalho. As regras que forem satisfeitas, serão selecionadas e ordenadas em uma lista através do método de resolução de conflito selecionado pelo usuário. Os padrões direitos associados das regras serão combinados com as instâncias resultantes da comparação e utilizados como novos fragmentos de conhecimento. Neste tipo de estratégia o sistema especialista irá parar quando não houver mais regras a executar ou após um número pré-determinado de ciclos. O encadeamento progressivo é adequado a problemas orientados a dados.

- **Regressivo** (para trás).Na estratégia regressiva, para trás, o problema consiste em um objetivo a ser atingido. Armazena-se o objetivo na memória de trabalho e comparam-se os padrões direitos das regras com o objetivo. As regras cujas conclusões corresponderem ao objetivo, serão selecionadas e ordenadas segundo o método de resolução de conflito adotado. Os padrões esquerdos das regras selecionadas serão armazenados na Memória de Trabalho como novos objetivos. Também neste caso o sistema especialista finalizará quando não houver mais regras a executar ou após atingir um número de ciclos previamente determinado. As aplicações típicas deste tipo de estratégias são os problemas de diagnósticos e detecção de falhas

Estas expressões vem do inglês "forward"e "backward". Cabe ao usuário escolher uma destas estratégias. Existe ainda a possibilidade de utilizar um encadeamento misto, entretanto esta estratégia deve ser programada diretamente pelo usuário.

Representação de Incerteza

O *FASE* permite atribuir aos fatos e regras uma medida numérica que representa de alguma forma a confiança do especialista. Estão disponíveis no *FASE* os seguntes métodos para a representação de incerteza:

- Fator de Certeza (conforme o modelo adotado no MYCIN) [Sho76]
- Teoria de Possibilidades [DP88]

Representação do Conhecimento

O módulo de representação de conhecimento consiste em uma base de conhecimento interna e uma linguagem de acesso que implementa um formalismo de representação de conhecimento. Encontram-se disponíveis no FASE três tipos de formalismo (ver seção 1.5):

- Lógica,
- Quadros
- Rede Semânticas.

Cada formalismo é definido por uma linguagem formal, um mecanismo de raciocínio e uma interface entre o formalismo e a base de regras.

1.6.3 Regras e Representação do Conhecimento

No FASE cada regra possui a seguinte sintaxe:

```
Se <padrão esquerdo 1> .... <padrão esquerdo n> com <filtro> então
   <padrão direito 1> .... <padrão direito n>
```

onde cada <padrão esquerdo i > é interpretado como uma pergunta em uma das três linguagens de representação de conhecimento disponíveis no FASE, <filtro> é qualquer função LISP contendo as variaáveis da substituição que, quando avaliadas, com as respectívas váriaveis substituidas por valores reais, pode ser utilizada como um filtro de substituição, retornando T ou NIL. O <padrão direito 1 > é interpretado como um comando de armazenamento para uma das linguagens de representação de conhecimento disponível.

A linguagem de acesso de conhecimento associada a cada linguagem de representação de conhecimento é composta de uma expressão de armazenamento e uma expressão de <questionamento>.

A expressão de *"armazenamento"* é utilizada para introduzir um novo fato na base de conhecimento ou para eliminar um fato anteriormente armazenado pela expressão

<armazenar> precedida da expressão com a palavra reservada *off*.

A expressão *"questionar"* é utilizada para fazer uma pergunta à base de conhecimento. Esta pergunta geralmente contem variáveis e a resposta é sempre uma lista de substituições, onde cada substituição corresponde a uma instância do padrão perguntado, presente na base de conhecimento.

A sintaxe da linguagem de acesso é a seguinte:

- **Lógica**

```
<padrão> ::= (logic <questionar>+) | (logic <armazenar>+)

<armazenar> ::= (<predicado> <termo fechado>*)
                (off (<predicado> <termo fechado>*))

<termo fechado> ::= <constante> | (<função> <termo fechado>*)

<questionar> ::= (<predicado> <termo>*)

 <termo> ::= <constante> | <variável> | (<função> <termo>*)
```

- **Redes Semânticas**

```
<padrão> ::= (snet <questionar>+) | (snet <armazenar>+)

<armazenar> ::= (<nodo>) |
                (<nodo> <nodo> [<arco>]) |
                (<nodo> <nodo> [(not . <arco>)]) |
                (off (<nodo>)) |
                (off (<nodo> <nodo> [ <arco> ])) |
                (off (<nodo> <nodo> [ (not . <arco>) ]))


<questionar> ::= (<nodo> <nodo> [ <arco> ])

                  <nodo>  ::= <variável> | <nodo>

                  <arco>  ::= <variável> | <arco>  (not . <variável>)
                                                   (not . <arco>)
```

- Quadros

```
<padrão> ::= (frame <questionar>+) | (frame <armazenar>+)

<armazenar> ::= (<quadro> <quadro antecedente>
                    <definição de atributo>*) |
                    (off (<quadro>)) |
                    (off (<quadro> (<atributo>)+))

<definição de atributo> ::= (<atributo> <valor>) |
                     (<atributo>
                            <definição de função>
                             <demon type>) |

<demon type> ::= se-criar | se-modificar |
                 se-necessario | se-apagar

<valor> ::= <quadro> |
                   qualquer S-expression

<questionar> ::=  (<frame> [ <antecedente> ]
                       (<questionar atributo>*))

<frame> ::= <variável> |
                    <quadro>

<antecedente> ::= <variável> |
                         <quadro>

<questionar atributo> ::= (<atributo> <valor>)

<atributo> ::= < variável> |
                        <atributo>

<valor> ::= < variável> |
                    <quadro> |
                      qualquer S-expression
```

O FASE pode incorporar facilmente qualquer novo formalismo de representação de conhecimento que possa ser implementado de acordo com a metodologia questionamento/armazenamento descrita anteriormente.

1.6.4 Como Funciona o FASE.

O funcionamento do arcabouço de sistemas especialistas, construído a partir das bibliotecas do FASE pode ser descrito de forma simplificada, pelo seguinte procedimento:

1 Para cada regra :

[1.1] Selecionar o lado esquerdo ou o lado direito, de acordo com o tipo de estratégia de controle escolhido.

[1.2] Utilizar cada padrão do lado escolhido como uma pergunta à memória de trabalho, de acordo com o tipo de representação de conhecimento associado ao padrão, obtendo como resposta uma lista de instâncias.

[1.3] Caso todos os padrões resultem em listas de instâncias não vazias realiza-se a combinação destas listas obtendo uma lista única. Associa-se esta lista de instâncias à regra em questão.

2 Utilizando as listas de instâncias associadas às regras no passo anterior, escolhem-se quais regras serão realmente executadas e em que ordem, de acordo com o método de resolução de conflito escolhido.

3 Para cada regra a ser executada:

[3.1] Calculam-se os valores de incerteza das diversas instâncias da regra, de acordo com o tipo de tratamento de incerteza escolhido.

[3.2] Combinar cada instância resultante aos padrões presentes no lado oposto da regra e utilizar estas combinações como novos fragmentos de conhecimento a serem armazenados na memória de trabalho de acordo com os tipos de representação de conhecimento associados aos padrões.

Capítulo 2

Inteligência Artificial Distribuída

2.1 Introdução

A Inteligência Artificial Distribuída *(IAD)* pode ser vista como uma busca distribuída num espaço de estados [Les94]. Uma busca distribuída envolve o particionamento do espaço de estados e seus operadores associados, e um regime de controle permitindo que os diversos elementos de processamento possam realizar simultaneamente buscas locais em diferentes partes do espaço de estados. O resultado dessas buscas locais é compartilhado de forma tal que a resposta desejada seja produzida oportunamente. Uma busca distribuída no espaço de estados pode requerer estratégias diferentes daquelas consideradas ótimas em uma busca seqüencial.

As pesquisas em *IAD* enfatizam o desenvolvimento de técnicas para o particionamento do espaço de busca em conjuntos de busca locais e a coordenação dessas buscas simultâneas, tanto a nível de compartilhamento de informação quanto a nível de controle. A forma desse particionamento, a dimensão dessas novas buscas no espaço de estados, as características do processo que implementa tais buscas distribuídas, a distribuição da perícia e da informação na organização dos processos impulsionaram diferentes linhas de pesquisa em *IAD*.

Um dos primeiros ramos de pesquisa em *IAD* objetivou o estudo da decomposição da busca no espaço de estados em um conjunto de buscas distribuídas de fina granulação. Neste caso, o espaço de estado de busca é particionado em espaços de busca locais compostos por uma pequena coleção de estados, onde os operadores e o controle são implementados com baixo esforço computacional [Les94]. Essa busca distribuída de fina granulação pode ser obtida através de um conjunto de processos, conforme a nomenclatura da *IAD*, uma comunidade de agentes de granulação fina onde cada busca local é associada a um agente. Os agentes são de baixa complexidade, podendo existir vários em uma mesma organização. Tais agentes são interdependentes e se comunicam através de padrões pré-determinados com os agentes vizinhos. Os pesquisadores que

se dedicaram a esse ramo da *IAD* se autodenominaram *Conexionistas* e se afastaram dos demais pesquisadores de *IAD*, uma vez que seu interesse era bem diferente dos demais.

Uma outra corrente que se formou na *IAD* preocupou-se em estudar a decomposição de alta granulação, onde a busca no espaço de estados é particionada em relativamente poucos conjuntos: conjunto de subproblemas dependentes e independentes. Nessa abordagem onde a distribuição da busca no espaço de estados é feita com uma granulação alta, a atividade de solucionar os problemas é realizada por um grupo de *agentes* de alta complexidade. Tais agentes utilizam estratégias sofisticadas para coordenar a solução do problema em questão e para se comunicar com outros agentes. A maior parte das pesquisas em *IAD* concentrou-se na decomposição do espaço de busca em sociedades de agentes de alta granulação. Esse modelo de distribuição em *IAD*, passou a se chamar *Solução Distribuída de Problemas (SDP)* [Les94].

A *SDP* teve como preocupação inicial, a construção de sistemas distribuídos para solucionar os problemas de Inteligência Artificial de alta complexidade ou de natureza distribuída. A maior parte dos trabalhos em *SDP* tem se concentrado em explorar a robustez e os recursos disponíveis das redes de computadores para solução desses problemas, como por exemplo, a abordagem utilizada pelo Contract-Net [Smi80] para a decomposição e alocação de tarefas em uma rede de computadores. Uma outra linha de trabalhos bastante presente na *SDP* é a utilização de múltiplas fontes de conhecimento para solução de problemas, como por exemplo controle de trafego aéreo, monitoramento de demanda de veículos, monitoramento ambiental, etc.

A *SDP* concentrou seus esforços no problema a ser resolvido, tomando como premissa que os agentes da *SDP* estariam sempre dispostos a cooperar, compartilhar informações e comunicar-se de forma confiável. Entretanto, experiências em ciências sociais tem deixado claro que atingir tais propriedades em grupos de indivíduos está longe de ser uma tarefa trivial [Bit96]. A constatação de que os agentes computacionais de uma dada comunidade não podem manter uma total dependência dos demais agentes da comunidade, no que diz respeito a: compartilhamento de informações, concordância e disponibilidade para cooperar, deu origem a uma nova linha de pesquisas em *IAD* chamada *Sistemas Multi-Agentes, SMA* [DR94].

A linha de *SMA* mantém o foco dos seus trabalhos no agente, mais precisamente nas propriedades do *agente* e nas premissas básicas que assegurem um comportamento cooperativo da comunidade de agentes computacionais. O estudo em *SMA* possui um caráter naturalmente interdisciplinar envolvendo conceitos oriundos de diversas disciplinas, como por exemplo, Economia, Teoria de Jogos, Ciências Sociais, Etologia, etc. Dentre esses conceitos o de *Agente Racional*, aquele que age no sentido de maximizar seus benefícios e minimizar suas perdas, apresenta um especial interesse para a área de *SMA*.

A seguir, os principais conceitos desenvolvidos pela *IAD* são apresentandos. Em seguida os dois principais ramos de pesquisa atualmente em desenvolvimento são ap-

resentados: *SDP e SMA*. Por fim são apresentados tres óticas relacionando esses dois ramos da *IAD : SDP e SMA* [DR94].

2.2 Conceitos Básicos

A *IAD*, um ramo recente da IA que tem apresentado um enorme crescimento nos últimos anos, estuda o conhecimento e as técnicas de raciocínio destinadas a permitir o agrupamento de *agentes computacionais* em *sociedades de agentes*, caracterizadas por demonstrar um comportamento inteligente. Entende-se por *agente* toda entidade de um dado ambiente capaz de: detectar as transformações ocorridas no meio onde esta inserido e de atuar sobre esse meio. Uma *sociedade de agentes* é formada por um grupo de agentes que possuem objetivos comuns. A *IAD* apresenta cinco questões básicas, interrelacionadas, as quais têm se mostrado presentes em grande parte dos trabalhos apresentados:

- **Descrição, decomposição e alocação de tarefas:** Como descrever e decompor facilmente uma tarefa complexa em subtarefas, como estas subtarefas serão alocadas e como selecionar o agente, ou os agentes, mais apropriados para realizar as respectivas tarefas.

- **Interação, comunicação e linguagens:** Quais primitivas devem ser utilizadas por um protocolo de comunicação de maneira a expressar a semântica dos conceitos envolvidos em um trabalho cooperativo.

- **Coordenação, controle e sociabilidade:** Como assegurar um comportamento global coerente, em uma sociedade composta por diversos agentes, cada um com suas respectivas habilidades e objetivos e como projetar uma estratégia de controle adequada.

- **Conflitos e incertezas:** Uma vez que nenhum dos agentes da comunidade possui um conhecimento global a respeito do ambiente, como solucionar os conflitos que venham a surgir e como tratar a incerteza; e a falta de informação garantindo resultados coerentes.

- **Linguagens de programação e ambientes:** Do ponto de vista computacional, quais as linguagens de programação mais adequadas à construção de tais sistemas; e quais são os requisitos para os ambientes destinados a construção desses sistemas.

A *IAD* se mostra bastante apropriada para a próxima geração de aplicações e organizações computacionais, atualmente em fase de planejamento. Estas aplicações serão estruturalmente complexas, comunidade de agentes capazes de suportar a participação de agentes computacionais e agentes humanos. Essa próxima geração de sociedade de agentes vai propiciar aos pesquisadores de *IAD* novos desafios, uma vez que tais sociedades apresentarão um caráter bastante heterogêneo. Elas necessitarão atender tanto a requisitos temporais flexíveis quanto a rígidos, deverão apresentar robustez e permitir um grande número de participantes. Atualmente existem diversas linhas de pesquisa e metodologias de implementação, como por exemplo a criação de aplicações do

mundo real que integrem os conhecimentos já adquiridos e novas teorias e implementações práticas voltadas a essa nova geração de aplicações [DR94].

2.3 Solução Distribuída de Problemas

A *SDP* é geralmente definida como um sistema distribuído fracamente acoplado, composto por agentes semi-autônomos que realizam sofisticadas técnicas de solução de problemas e que cooperativamente interagem com outros agentes para solucionar *um único problema.* Estes agentes atingem seu objetivo através da solução individual de subproblemas e da integração das respectivas soluções dos subproblemas em uma solução global do problema em questão (Esses que podem ser interdependentes ou sobrepostos). A solução global não necessariamente requer a participação de todos os agentes da sociedade. Em certas situações os participantes de uma dada solução podem estar distribuídos ao longo de uma rede de computadores, integrados apenas pela sua consistência mútua. Devido à distribuição do conhecimento e da informação na rede, freqüentemente um dos agentes da sociedade se vê impossibilitado de apresentar uma solução sem a assistência de um outro agente. Em virtude dos limites impostos pelos canais de comunicação, os agentes adotam a estratégia de fazer o melhor possível com o conhecimento disponível. Os agentes cooperam, através da geração e do intercâmbio de tentativas e resultados parciais, com os demais participantes do grupo, até que a informação compartilhada seja suficiente para a construção da consistência mutua, completando a solução dos subproblemas locais.

Na *SDP*, cada um dos *agentes* participantes pode modificar seu comportamento em função de mudanças circunstanciais além de poder planejar suas próprias estratégias de comunicação e cooperação com os demais participantes. Cada um dos participantes do grupo trabalha de forma assíncrona, paralelamente aos outros participantes e sujeito a uma comunicação limitada como os demais participantes.

A *SDP* pode ser representada esquematicamente da seguinte maneira [SDB92]:

$$\textbf{problema} \Longrightarrow \textbf{projeto} \Longrightarrow \textbf{método de solucão} \Longrightarrow \textbf{resultado}$$

Existe uma diferença significativa entre *SDP* e Processamento Distribuído. O processamento distribuído em uma rede de computadores, tipicamente possui uma diversidade de tarefas sendo executadas concorrentemente na rede. Neste caso o acesso compartilhado aos recursos físicos ou à informação é o principal motivo para a interação entre as tarefas. O objetivo é manter a ilusão de que cada uma das tarefas está sendo executada isoladamente em um sistema dedicado, sendo o compartilhamento de recursos e gerenciamento de conflitos entre as tarefas realizados, de forma transparente para o usuário, pelo sistema operacional da rede. Por outro lado, os processos para solucionar os problemas em *SDP* trabalham juntos na solução de um mesmo problema. Tais processos são explicitamente informados a respeito da distribuição dos componentes na rede e podem interagir sobre informações e decisões baseadas nessas informações. Diferentemente de *SDP*, onde a cooperação entre seus participantes é crucial para o

desenvolvimento da solução, os processos nas aplicações tradicionais de processamento distribuído, raramente necessitam da assistência de um outro processo para executar sua tarefa.

Os pontos mais importantes na *SDP* são os seguintes:

- Uma vez que os agentes estão trabalhando cooperativamente, não há necessidade de representar explicitamente as habilidades e metas dos demais agentes da sociedade. Isto é implicitamente representado pelo projetista.
- A descrição e a decomposição das tarefas, na maioria dos casos, é decidida pelo projetista. Esse é um caso extremo, mas mesmo quando ocorre alguma decomposição de tarefa, os métodos utilizados são fortemente dependentes do domínio da aplicação.
- Na maioria dos casos, não acontecem situações conflitantes, pois a divisão das tarefas foi devidamente realizada pelo projetista. Mais uma vez isso é um caso crítico, mas se aparecerem conflitos, estes estão fortemente relacionados com o domínio da aplicação.
- Mesmo quando os agentes podem se comunicar livremente, não há necessidade de realizar diálogos complexos para se atingir a meta.
- Não é possível adicionar novos agentes dinamicamente na sociedade. Isso significa que uma *SDP* não pode ser considerada um sistema aberto.

A *SDP* tem se tornado uma importante linha de pesquisa em *IAD* por diversos motivos. Primeiramente, os avanços tecnológicos na construção e na comunicação entre processadores, tornaram possível conectar um grande número de sofisticadas unidades assíncronas de processamento. A utilização de processadores interconectados pode ser significativa para propiciar a capacidade computacional requeridas nas aplicações de IA. Uma possível extensão das estruturas conectadas para um forte acoplamento de processadores é realizada através do compartilhamento ou distribuição de memória. Um acoplamento moderado dos processadores através de uma rede de comunicação local e para um processamento geometricamente distribuído fracamente acoplado através de uma rede de comunicação de longa distância.

Em segundo lugar existem diversas aplicações em *IA* que são inerentemente distribuídas. Essa aplicações podem ser espacialmente distribuídas, como a integração e interpretação de sensores espacialmente distribuídos ou o controle de um conjunto de robôs que trabalham conjuntamente em uma instalação fabril. As aplicações podem ser distribuídas segundo um critério funcional, como por exemplo agrupar sistemas de diagnóstico médico de diferentes especialidades para analisar um caso complicado, ou ainda um sofisticado sistema especialista para projetos arquitetônicos compostos por especialistas em projetos estruturais, elétricos, hidráulicos, "layout" etc. As aplicações podem ainda ser distribuídas segundo o critério temporal, como em uma aplicação fabril onde existe uma linha de produção composta por um certo número de áreas de trabalho nas quais um sistema especialista é responsável pelo escalonamento dos

pedidos. Uma arquitetura para solução de problemas que contemple a distribuição de dados, conhecimento e capacidade de processamento tem vantagens significantes sobre uma simples, monolítica e de processamento centralizado, no que diz respeito a: processamento e eficiência na comunicação, confiabilidade, robustez e resposta em tempo real.

A habilidade para estruturar um problema complexo em módulos de processamento "auto-contidos", permite a implementação de sistemas mais fáceis de construir, depurar e manter do que um módulo único e monolítico, além de permitir uma tolerância a erros tanto em "software" como em "hardware".

Finalmente, o entendimento do processo de *SDP* é uma importante meta de *IA*. O desenvolvimento de sistemas para *SDP* pode ser utilizado para validar teorias em sociologia, gerenciamento e teoria organizacional, bem como desenvolvimento de sistemas em *IA* para teoria de soluções de problemas, lingüística, psicologia e filosofia.

2.4 Sistemas Multi-Agentes

Historicamente, os primeiros trabalhos de pesquisa em *IAD*, foram relacionados a *SDP*. A incorporação de alguns resultados práticos e técnicas provenientes das ciências sociais, psicologia e etologia, nesse domínio, deram origem aos *Sistemas Multi-Agentes, (SMA).*

As primeiras tentativas de solucionar problemas cooperativamente surgiram nos anos setenta [FG91]. Uma dessas primeiras tentativas foi o projeto HEARSAY-II [Erm80], um sistema para entendimento de diálogos que introduziu "O Modelo Quadro Negro" (do inglês "Blackboard Model" [HR85]). Na mesma época alguns modelos computacionais, como por exemplo a essência de Lenat [Len88] e os atores de Agha e Hewitt [AH88], foram desenvolvidos visando tratar problemas como compartilhamento de recursos, controles complexos e o aparecimento de comportamentos complexos provenientes de interações bastante simples.

Durante os anos oitenta, a importância de representar explicitamente no *agente*, um conhecimento sobre os demais agentes da sociedade, no contexto do planejamento, foi levantada por Konolige e Nilson [KN80]. A utilização de modelos organizacionais humanos, no trabalho cooperativo de *agentes* computacionais foi discutido por Fox [Fox81] e o "Contract Net Protocol", um protocolo de negociação baseado em uma estrutura de mercado, foi desenvolvido por Smith [Smi80].

A utilização de arquiteturas simples, e de pequeno porte, para projetar robôs autônomos, foi introduzida por Brooks, os chamados "Robôs Reativos" [Bro86]. O conceito de Sistemas Abertos [Hew88], introduzido por Hewitt, foi crucial para as pesquisas em *SMA*. Finalmente alguns dos primeiros ambientes para implementação e teste de algumas dessas idéias foram desenvolvidos, por exemplo DVMT [CL83], DPSK [Car87] e MACE [Gas87].

Os *Sistemas Multi-Agentes* baseiam-se na idéia de *Comunidade Inteligente* [SDB92], ou seja, a inteligência do sistema emerge do comportamento social da comunidade. Essa comunidade é formada por *agentes autônomos*, inseridos em um ambiente comum, capazes de cooperar para alcançarem uma meta global. Os membros de uma comunidade inteligente podem ser de baixa complexidade computacional, *Agentes Reativos* [Bro86], ou extremamente complexos *Agentes Cognitivos ou Inteligentes.* Os trabalhos de pesquisa realizados nessa área se concentram basicamente nas propriedades dos agentes, mais precisamente naquelas propriedades que o levam a cooperar com os demais membros da comunidade.

Os agentes, em uma abordagem *SMA*, coexistem em um ambiente comum e estes podem colaborar uns com os outros visando atingir metas globais. Este comportamento pode ser representado esquematicamente da seguinte maneira:

$$\textbf{agentes} \Longrightarrow \textbf{problema} \Longrightarrow \textbf{método de solucão} \Longrightarrow \textbf{resultado}$$

Os pontos mais importantes em *SMA* são:

- **A decomposição das tarefas:** realizada pelos agentes da sociedade, e não pelo projetista. Em casos extremos é possível realizar uma reorganização dinamicamente, ou seja, um determinado agente da comunidade pode decidir modificar seu comportamento visando otimizar a realização de suas tarefas;

- **Autonomia:** os agentes são autônomos, podendo ter suas próprias metas locais. Um determinado agente da comunidade, pode decidir não participar de um trabalho cooperativo caso esse agente julgue mais importante cumprir uma meta local. Consequentemente, torna-se comum o aparecimento de conflitos, causados pela existência de metas locais e metas globais;

- **Concorrência:** os agentes da comunidade estão habilitados a realizar outras tarefas cooperativas, ou não paralelamente desde que estes possuam as habilidades requeridas por estas tarefas;

- **Sistema Aberto:** um *SMA* é um sistema aberto, um agente pode entrar ou sair da sociedade a qualquer instante. Caso um novo agente ingresse na sociedade, suas habilidades e capacidades devem ser incorporadas ao *Comportamento Inteligente* da sociedade. Isto normalmente é realizado através da manutenção de uma representação explicita, em cada agente, a respeito das habilidades e metas dos demais agentes da comunidade.

- **Mudanças Ambientais:** o ambiente no qual a comunidade está inserida está sujeito a mudanças e os agentes da comunidade devem incorporar tais modificações em seus respectivos modelos ambientais. Um exemplo de agentes que necessitam interagir com ambientes dinâmicos é o caso de robôs móveis que navegam em espaços diferentes e desconhecidos.

Os *SMA* são um campo recente da *IAD*, entretanto, existe um número significativo de sistemas, já desenvolvidos, que utilizam estas técnicas. Podem ser citados trabalhos em processamento de linguagem natural [Ste92], construções inteligentes e sistemas tutoriais para aprendizado [CGR92]. No campo industrial, o projeto ESPRIT II utiliza o modelo ARCHON (Arquitecture for Co-operative Heterogeneos On-Line System) [OC91], está tentando integrar em um *SMA* alguns sistemas baseados em conhecimento já existentes.

2.5 Relações entre SDP e os SMA

A relação entre as áreas de *SDP e SMA* tem sofrido modificações ao longo do tempo, e ainda hoje não existe uma fronteira bem definida entre esses dois enfoques. A seguir são mostrados três pontos de vista, não mutuamente exclusivos, relacionando *SDP e MAS* [DR94].

2.5.1 SDP como um subconjunto da SMA

Uma visão que tem mantido sua consistência com as definições gerais adquiridas por *SMA* ao longo dos anos, *SDP* como um subconjunto de *SMA*. De fato um *SMA* pode ser considerado uma *SDP* quando certas hipóteses são asseguradas, dentre elas: benevolência, compartilhamento de objetivos globais, a existência de um projetista centralizador.

Benevolência: Uma das hipóteses que vem sendo proposta como um "divisor de águas" para classificar um dado sistema como uma *SDP* é se os agentes da comunidade são benevolentes. Um *agente* é dito benevolente quando este está sempre disposto a ajudar um outro agente da comunidade. Por exemplo, no "Contract Net protocol", os agentes distribuem as tarefas a serem realizadas baseados na adequação da tarefa ao *agente* e na disponibilidade deste, ignorando se os demais *agentes* irão realizar ou não tal tarefa. Quando um determinado *agente* necessitar alocar uma tarefa, ele abre uma licitação e difunde o edital desta licitação entre os agentes da sociedade. Uma vez percebido que um edital para realização de uma determinada tarefa foi difundido, os *agentes* dispostos a realizar tal tarefa, irão enviar, honestamente, uma proposta para realização da tarefa licitada. As propostas enviadas ao agente licitante possuem um indicador para expressar o quão bem a tarefa licitada pode ser realizada pelo agente. O agente, ou os agentes que apresentarem as melhores propostas realizam a tarefa. Os agentes da comunidade não precisam ser ou persuadidos a realizar uma determinada tarefa.

Mesmo com a hipótese da benevolência, a cooperação e um processo coerente de coordenação estão longe de serem asseguradas. Ainda que os *agentes* da comunidade desejem fazer o melhor que podem, dificuldades temporais e perspectivas locais podem levar a uma atividade não cooperativa e sem coordenação. No "Contract Net", por exemplo, tarefas importantes podem deixar de ser reivindicadas quando agentes adequados estão ocupados com outras tarefas que poderiam ser realizadas por outros agentes, ou ainda algumas tarefas podem ser designadas impropriamente levando os agentes a realizar

atividades redundantes ou incompatíveis. Concluindo, a benevolência apenas não é capaz de garantir um comportamento cooperativo.

Compartilhamento de Objetivos Globais A motivação para a benevolência em uma sociedade de agentes é a existência de objetivos globais. Realmente, se todos os agentes valorizarem o mesmo resultado de atividade global, eles irão tentar contribuir de qualquer maneira possível para esse resultado global. Esta hipótese pode ser considerada o "coração" do "Contract Net" [GI91]. No DVMT, por exemplo, os objetivos globais do sistema são atingidos solicitando aos agentes da sociedade que integrem seus respectivos mapas locais dos movimentos do veículo, no caso o agente, em um mapa global de movimentos dos veículos, a comunidade. Uma vez que todos os agentes estão tentando ajudar o sistema a convergir para uma solução global, cada um dos agentes está tentando ajudar os demais a obter uma boa interpretação local o mais rápido possível.

Uma visão parcial do problema a ser resolvido pode conduzir a decisões locais, que são globalmente incoerentes. A ausência de diretrizes bem definidas sobre responsabilidade ou interesses, pode levar os agentes a inundar uns aos outro com informações supérfluas. Mais ainda, os agentes podem enviar informações capazes de distrair os outros membros da sociedade.

Também obscuro é a prioridade que uma tarefa comum deve ter para tornar um sistema em uma *SDP*. Caso os agentes estejam concentrados para absorver uma determinada licitação, eles devem compartilhar um objetivo de maior prioridade, tanto para manter a licitação aberta enquanto os agentes possuem objetivos contrapostos como para determinar o vencedor da licitação. Da mesma forma, quando os agentes estão representando lados opostos em um processo licitatório, eles devem compartilhar um objetivo para poder chegar a um acordo, enquanto possuem preferências diversas a respeito do candidato à licitação racional.

Projetista Centralizador A hipótese da existência de um projetista centralizador para o sistema, a mais recente, tem colocado como característica de uma *SDP* a existência de um projetista centralizador. Essa perspectiva se sobrepõe às hipóteses anteriores, uma vez que o projeto de objetivos centralizados pode estar embutido no agente, e o projetista, preocupando-se que as subtarefas são, na verdade, parte dos objetivos globais, preferir tornar os agentes da comunidade benevolentes. Mais precisamente, uma vez que o projetista possui a visão global do sistema, as preferências dos *agentes* podem ser ajustadas e os mecanismos para expressão e atuação dessas preferências podem ser padronizados.

A questão aberta aqui, como no caso da hipótese objetivos globais, é: quais os detalhes que devem ser especificados pelo projetista, durante o projeto dos agentes, para torná-los parte de uma *SDP*?

Esta primeira visão que *SDP* é um subconjunto de *SMA*, possui argumentos favoráveis, entretanto, padece por algumas desvantagens. Primeiramente, não estabelece delimitação

clara entre os dois sistemas, *SDP e SMA*, principalmente no que se refere às hipóteses dos objetivos globais e da existência de um projetista centralizador. Uma segunda desvantagem apresentada nessa ótica é que os agentes na *SDP* podem apresentar um comportamento de propósitos cruzados, distraindo os demais agentes da comunidade e praticando ações não cooperativas. Neste caso um observador não estaria habilitado a dizer se o sistema é uma *SDP* ou um *SMA*, apenas observando o comportamento dos agentes da comunidade.

2.5.2 SMA fornece a base para SDP

Tradicionalmente, as pesquisas em *SDP* tomaram, como seu ponto de partida, que as propriedades do sistema podem ser assumidas na fase de projeto. Estas propriedades incluem que os *agentes* conseguiram se comunicar de forma confiável, que eles irão seguir os protocolos de comunicação definidos, que eles irão realizar as tarefas conforme prometido, que eles irão prometer executar as tarefas quando solicitados e quando estiverem habilitados, para tal realização.

Assumindo estas propriedades internas, *SDP* concentra seus esforços no desenvolvimento de técnicas que conduzam o sistema a demostrar certas propriedades desejadas. Tipicamente, estas propriedades externas geram as soluções apropriadas para as instâncias do problema que originalmente motivaram a construção do sistema. Estas instâncias podem envolver diferentes combinações entre tarefas e o ambiente, incluindo a distribuição inicial das tarefas, tempo de chegada das tarefas, falhas de comunicação entre os agentes da sociedade, atrasos de comunicação, etc. Por exemplo, no sistema DVMT, a propriedade do sistema que sofreu o maior monitoramento foi o tempo de resposta, no caso, quanto tempo era necessário para a comunidade de agentes gerar um hipótese correta dos movimentos de um veículo na área em questão. A aceitação de uma estratégia de coordenação foi considerada bem aceita, no caso dos agentes da comunidade conseguirem realizar o rastreamento dos veículos com boas respostas temporais, apesar da perda de algumas mensagens, da varredura dos dados de entrada e as falhas apresentadas por alguns agentes.

Enquanto a *SDP*, em geral, admite que as propriedades internas desejadas do sistema podem ser mantidas, os *SMA*, geralmente, estão preocupados, em primeiro lugar, em assegurar tais propriedades. De fato, os *SMA* geralmente realizam hipóteses apenas sobre as propriedades individuais dos agentes, e consideram que tais propriedades irão emergir internamente dentre os agentes determinando os incentivos e características do ambiente. As atividades de pesquisa em *SMA* podem definir estruturas de incentivos ou características ambientais que existam naturalmente ou que possam ser impostas, assegurando que as propriedades internas desejadas sejam alcançadas.

Esta visão da *IAD* apresenta uma divisão entre as principais abordagens em vigor, *SDP e SMA*. *SMA* estuda como os indivíduos auto-interessados em suas tomadas de decisão podem descobrir, ou ser coagidos a atribuições estáveis e estratégias desejadas para interagir com os demais agentes da comunidade. *SDP* por sua vez, considera

como esta dependência das interações desejadas podem ser inicializadas, controladas e explorados de maneira a produzir um sistema que execute alguns objetivos externamente definidos.

2.5.3 SMA e SDP como linhas de pesquisa complementares

A segunda visão proposta para relacionar *SMA e SDP*, onde se acredita que SMA fornece a base para SDP, mostrou, ainda que implicitamente, que as questões investigadas pelos pesquisadores de *SMA* são diferentes das investigadas pelos pesquisadores de *SDP*. Isto mostra que *SMA e SDP* não são apenas nomes de sistemas computacionais e sim linhas de pesquisa diferentes. Como já foi mencionado, existe a possibilidade de um observador, não conseguir classificar um dado sistema como *SMA ou SDP* apenas observando o seu comportamento. Entretanto um sistema pode ser parte de uma linha de pesquisa.

A título de exemplo, considere o problema de planejamento e execução realizado por múltiplos agentes de forma descentralizada. Diversos sistemas tem sido desenvolvidos para este fim, apresentando muitas características em comum. Entretanto o que os torna parte de diferentes objetos de pesquisa são exatamente os tipos de questões respondidas pelos pesquisadores, desenvolvendo, analisando e realizando experiências com tais sistemas.

Na abordagem do planejamento parcial, por exemplo, os agentes realizam seu planejamento local dinamicamente em resposta às mudanças ocorridas em seus respectivos domínios no ambiente e as mudanças a respeito das intenções dos demais agentes. Sendo assim, em qualquer instante, um dado agente terá um modelo das intenções coletivas de um subconjunto dos agentes da comunidade, o chamado planejamento parcial, e modificará suas atividades apropriadamente, assumindo que os demais agentes irão modificar suas atividades de forma compatível. Esse trabalho de pesquisa focaliza os seguintes aspectos: a eficiência dos agentes na coordenação de seus planos, a robustez da sociedade de agentes em suportar falhas da rede de computadores ou dos agentes, o impacto dos atrasos e perdas das mensagens sobre a performance da sociedade de agentes [DR94].

O trabalho realizado por Ephrati e Rosenschein visa determinar "como podem os agentes convergir para um planejamento para atingir objetivos comuns quando cada um destes está construindo uma parte desse planejamento localmente". Assim como no planejamento parcial, essa abordagem envolve uma troca de informações sobre os aspectos dos planejamentos locais e a integração dessa informação para identificar as relações que podem levar a mudanças nestes planejamentos locais. Mas, até então, as linhas da pesquisa divergem do planejamento parcial. As grandes questões respondidas por Ephrati e Rosenschein foram sobre a extensão para que a abordagem deles realizasse apropriadamente em face da manipulação, agentes não sinceros e sobre o desconhecimento do ambiente quanto às falhas e atrasos na comunicação.

	Propriedades		
	Agentes	Ambiente	Sistema
SMA	variável	fixa	fixa (interna)
SDP	fixa	variável	fixa (externa)
?	fixa	fixa	variável

Tabela 2.1: Temas de Pesquisa em IAD

Em contraste com a visão anteriormente apresentada em que SMA fornece a base para SDP, essa terceira visão sobre *SMA e SDP* as enxerga como linhas de pesquisa que partiram de um ponto comum mas exploram diferentes aspectos. Digamos que a *IAD* envolve *agentes* que atuam em um *ambiente* para compor um *sistema.* Podemos então falar a respeito das propriedades dos *agentes*, do *ambiente* e do *sistema.*

- **Propriedades dos Agentes:** O que podemos dizer a respeito de um agente individualmente? Ele é racional? Suas preferências fazem parte de um conhecimento comum? Eles são compartilhados? Quais são suas aptidões? Eles se conhecem uns aos outros?

- **Propriedades do Ambiente:** O que podemos dizer a respeito do ambiente? Ele é estático? Fechado? Aberto? As ações realizadas pelos agentes são previsíveis? Limitadas no Tempo?

- **Propriedades do Sistema:** O que podemos afirmar sobre o conjunto: agente, ambiente e sistema? O sistema pode assegurar certas propriedades internas, tais como um acesso justo aos recursos ou a honestidade entre os agentes? Ele pode assegurar certas propriedades externas, tais como a correção temporal?

A partir desses três tipos de propriedades pode-se resumir *SMA e SDP* como está apresentado na tabela 2.1.

Ou ainda:

- **SMA:** Corresponde a uma linha de pesquisa que se preocupa com certas propriedades internas em um sistema composto por agentes cujas propriedades individuais podem variar. Realmente, *SMA* tem se preocupado como agentes com preferências individuais irão interagir em um dado ambiente de forma a levar o sistema a apresentar as propriedades globais desejadas. *SMA* responde como, para um dado ambiente, podem certas propriedades coletivas do sistema ser realizadas, uma vez que as propriedades dos agentes podem variar livremente.

- **SPD:** Tem concentrado seus esforços na obtenção das propriedades externas tais como, robustez, eficiência, sob uma variedade de condições ambientais, estabelecendo-se as propriedades dos agentes. *SDP* responde como é possível uma particular colocação de agentes alcançar alguns níveis de atuação coletiva caso as propriedades do ambiente estejam dinamicamente descontroladas.

A maioria dos sistemas não vai poder ser classificada inteiramente em um desses dois enfoques da *IAD, SMA ou SDP*. Da mesma forma os trabalhos de pesquisa em *IAD* apresentaram características de ambos os enfoques. Finalmente existe uma questão a ser respondida pela comunidade de pesquisadores em *IAD*: "Como se deve chamar as linhas de pesquisas em *IAD*, onde as propriedades dos agentes e do ambiente são fixas, mas as propriedades do sistema variam?" É possível até especular que alguns trabalhos em vida artificial ou redes neurais possam se enquadrar nessa categoria, mas isso merece um estudo mais aprofundado.

Capítulo 3

Sistemas Multi-Agentes

3.1 Introdução

Atualmente não existe uma definição para *agente* aceita por toda a comunidade de *IAD*. Entretanto uma definição genérica foi proposta por Ferber [FG91].

"Um *agente* é uma entidade real, ou virtual, emergido em um dado ambiente onde ele pode tomar algumas ações, estar habilitado para perceber e representar parcialmente este ambiente, pode ainda comunicar-se como os demais agentes do ambiente. Este *agente* apresenta um comportamento autônomo que é uma conseqüência de: suas observações, do conhecimento armazenado e das interações com os demais agentes do ambiente."

Os *SMA* baseiam-se na idéia de *Comunidade Inteligente* [Bit96], ou seja, a inteligência do sistema emerge do comportamento social da comunidade. Essa comunidade é formada por *agentes autônomos*, inseridos em um ambiente comum, capazes de cooperar para alcançarem uma meta global. Os membros de uma comunidade inteligente podem ser de baixa complexidade computacional, *Agentes Reativos*, ou extremamente complexos *Agentes Cognitivos ou Inteligentes*. Os trabalhos de pesquisa realizados nessa área se concentram basicamente nas propriedades dos agentes, mais precisamente naquelas propriedades que o levam a cooperar com os demais membros da comunidade.

3.2 Agentes Reativos

Os *Agentes Reativos* são baseados em modelos de organização biológicas ou etológica, como por exemplo as colméias, as sociedades de formigas ou cupins. Embora o comportamento de uma formiga isoladamente não possa ser considerado inteligente, o formigueiro é um exemplo claro de uma *Comunidade Inteligente*, uma vez que existe uma busca de alimentos e posterior estocagem, a reprodução é organizada apresentando berçários, enfermeiras, etc.

Os *Agentes Reativos* apresentam, de certa forma, alguma similaridade com a abordagem conexionista da *IA*. Entretanto, uma vez treinada a rede neural, não é possível modificar o seu comportamento. Os *Agentes Reativos*, por sua vez, são capazes de acompanhar as modificações do ambiente.

O modelo de funcionamento de um *Agente Reativo* é o de *Estímulo-Resposta*. Geralmente estes *agentes* não apresentam uma memória das ações tomadas e também não planejam suas ações futuras. Todo o conhecimento a respeito das ações e do comportamento dos demais *membros da sociedade* é percebido através das modificações sofridas pelo ambiente. Estes *agentes* não possuem um modelo de comunicação de alto nível, nem uma representação explícita do ambiente ou dos membros da sociedade.

As *sociedades de agentes reativos*, em geral, são numerosas, apresentando milhares de membros em uma mesma comunidade. Existem diversos trabalhos na literatura sobre *SMA* baseados em agentes reativos, por exemplo: a arquitetura de subsumção de Brooks [Bro86], o modelo PACO [Dem93], entre outros.

3.3 Agentes Cognitivos

Os *agentes cognitivos* são baseados nos modelos de *organizações sociais*, em termos de sociedades humanas : grupos, hierarquias, mercados, etc. Estes *Agentes Cognitivos* possuem uma representação explícita do ambiente e dos membros da *comunidade* e podem raciocinar sobre as ações tomadas no passado e planejar as futuras ações. Os *Agentes Cognitivos* podem ainda interagir com os demais membros da comunidade através de: linguagens e protocolos de comunicação complexos, sofisticadas estratégias de negociação, etc. Estes *agentes* normalmente apresentam elevada complexidade computacional e caracterizam-se por apresentar um comportamento inteligente tanto em uma *Comunidade de Agentes* como isoladamente. Estas comunidades geralmente são compostas por um pequeno número de participantes.

Existem diferentes tipos de *Sistemas Multi-Agentes Cognitivos*, não mutuamente exclusivos, a maioria deles diferem quanto à sua arquitetura, possibilidades de comunicação e complexidade do *agente*. Os quatro tipos de arquitetura de *Sistemas Multi-Agentes Cognitivos* geralmente utilizados são: *"Blackboard System"*, *Sistemas Multi-Agentes Federados*, *Sistemas Multi-Agentes Democráticos*, *Sistemas Multi-Agentes Abertos*.

- **"BlackBoard System:"** Consiste em um grupo de especialistas, chamados de *Fontes de Conhecimento*, interagindo através de uma memória compartilhada, chamada de *"Blackboard"*, onde todo o ambiente está representado. Quando um *agente* deseja realizar uma tarefa que não faz parte das suas habilidades, ele coloca esta tarefa no *"Blackboard"* onde um outro *agente*, com as habilidades requeridas, possa ler esta tarefa, processá-la e colocar o resultado de volta no *"Blackboard"*. Uma vez os resultados disponíveis no *"Blackboard"*, o *agente* originalmente interessado pode resgatá-los. A principal vantagem dessa abordagem é o fato dos *agentes* não precisarem conhecer uns aos outros, toda as informações necessárias estão disponíveis no *"Blackboard"*. O grande problema é a excessiva troca de mensagens realizada pelo suporte de comunicação para assegurar que todos os *agentes* possuem a mesma percepção do ambiente.

- **Sistemas Multi-Agentes Federados:** Um *Sistema Multi-Agentes Cognitivo* é chamado *Federado* quando existem *agentes* especiais, complexos, chamados de *Facilitadores*, que possuêm um conhecimento a respeito das habilidades de outro *agentes* mais simples. O papel dos *Facilitadores* é organizar o trabalho entre os *agentes* mais simples. Uma vez que uma determinada requisição é recebida por um *Facilitador*, ele se responsabiliza por encontrar um *agente* mais simples capaz de realizar esta requisição. A principal vantagem dessa abordagem é permitir um gerenciamento eficiente da comunicação entre os *agentes*. Por outro lado, a centralização de importantes tarefas torna o sistema como um todo sensível ao mau funcionamento do *agente facilitador*.

- **Sistemas Multi-Agentes Democráticos:** A principal característica de um *Sistema Multi-Agentes Democrático* é fato de todos os *agentes* da comunidade possuirem o mesmo nível hierárquico. Nesta abordagem, os *agentes* realizam ações coletivas assincronamente e a comunicação é uma destas ações. Esta comunicação tipicamente obedece às regras de alguma *Linguagem de Cooperação* adequada, como por exemplo *KQML ("Knowledge Query Manipulation Language")* [FLM95], uma linguagem que utiliza primitivas especiais para expressar crenças, necessidades e modalidades de comunicação. Um outro exemplo de *Linguagem de Cooperação*, *CooL ("Cooperation Language")* [BF94], baseada no "Contract Net Protocol" [GI91] e que suporta a comunicação de *agentes* utilizando um conjunto de mensagens chamado *Primitivas de Cooperação*. As principais vantagens dessa abordagem são a modularidade e a flexibilidade, e a desvantagem é que cada um dos *agentes* necessita conhecer, ao menos parcialmente, a identidade e as habilidades dos demais *agentes da comunidade*.

- **Sistemas Multi-Agentes Abertos:** Um *SMA* é dito *aberto*, quando a composição da comunidade não é fixa, permitindo que os *agentes* possam subscrever-se e desligar-se da comunidade, dinamicamente. Neste tipo de sistema, alguns serviços podem estar disponíveis ou não, de acordo com a composição da comunidade em um dado momento. Os *agentes* podem adaptar-se e escolher diferentes objetivos a serem alcançados, planos de execução ou padrões de cooperação, dependendo dos serviços disponíveis na comunidade. A principal vantagem dessa abordagem é a sua robustez.

A desvantagem é a excessiva troca de mensagens realizada por um complexo protocolo de comunicação capaz de assegurar o *Caráter de Comunidade Aberta.*

3.3.1 Comportamento Social

O comportamento dos *Agentes Cognitivos* pode ser classificado segundo dois critérios: a alocação das tarefas, locais ou globais, e a habilitação para realização das tarefas.

- **A alocação das tarefas:** Uma tarefa global envolve todos os agentes da comunidade, a tarefa local envolve apenas um *agente*;
- **habilitação:** Um determinado *agente* está habilitado a executar uma tarefa se ele possuir as aptidões necessárias para a sua realização;

Segundo estes critérios os *Agentes Cognitivos* podem apresentar quatro tipos de comportamento quando inseridos em uma *Sociedade de Agentes*: coabitação, cooperação, colaboração, distribuição.

- **coabitação:** Um *agente* pode e deve realizar suas tarefas locais. O simples fato de estar imerso em um ambiente comum não o obriga a cooperar com os demais *agentes da comunidade*;
- **cooperação:** Caso um *agente* esteja impossibilitado de realizar uma tarefa local, quer seja porque este não possui as aptidões necessárias ou por não conseguir realizá-la com a desenvoltura requerida, ele solicita aos demais agentes que cooperem com ele;
- **colaboração:** Um dado *agente* pode ser capaz de executar uma tarefa global sozinho. Caso existam mais *agentes* na comunidade capazes de realizar essa mesma tarefa, um mecanismo eletivo deve ser acionado;
- **distribuição:** Algumas tarefas globais precisam ser realizas por um conjunto de ações coletivas. Isto ocorre quando nenhum dos *agentes* podem alcançar o objetivo isoladamente, necessitando a divisão e alocação da tarefa segundo algum critério;

3.3.2 Descritores Internos e Externos

O conhecimento que um *agente* possui a respeito da suas aptidões e das aptidões dos demais membros da *comunidade* é crucial para que a *comunidade de agentes* apresente um comportamento *cooperativo / distribuído.* Este conhecimento é armazenado em duas estruturas de representação diferentes: os descritores Internos e Externos.

- **Internos:** Contém as aptidões e comportamentos do próprio *agente.* Esta descrição pode ser explícita, possibilitando ao *agente* conhecer seus procedimentos internos, ou não.
- **Externos:** Armazena as aptidões e comportamentos dos demais *agentes da comunidade.* Cada um dos *agentes* deve ter uma *representação externa* dos demais.

Basicamente, a *informação estática*, como por exemplo o conhecimento, e a *informação dinâmica*, o comprometimento com uma atividade para solucionar um problema, são representados nestes descritores. Um aspecto importante é que dois *agentes* podem ter o mesmo *descritor externo*, mesmo que possuam organização interna distintas. O conteúdo deste *descritor externo* é um tópico de pesquisa ainda aberto em *IAD*, estando intimamente relacionado com os protocolos de comunicação utilizados pelos *agentes* durante os processos de interação.

3.3.3 Protocolos de Comunicação

Os trabalhos de pesquisa em *IAD* dão ênfase à importância das ações e interações entre *agentes de uma comunidade*. A interface dos *agentes* como o ambiente pode ser dividida em: *percepção* e *comunicação*.

- **Percepção:** é a capacidade instintiva de um *agente* a respeito do ambiente, como por exemplo enxergar, ouvir, sentir, etc. Isto pode ser obtido em um robô, por exemplo, através de câmeras, microfones, sensores de pressão, etc.
- **Comunicação:** é troca de dados e conhecimentos entre os *agentes*.

A alta complexidade dessa comunicação obriga a definição de protocolos para expressar os conceitos semânticos envolvidos na solução cooperativa de problemas. Uma proposta adotada por vários pesquisadores é a utilização da ação comentada [CD90]. Resumidamente, esta teoria propõe que cada elocução seja vista como um tipo de ação, similar às ações externadas ao ambiente. Realmente, cada ação comentada carrega algum conteúdo proposicional mais uma força de locução, como por exemplo a solicitação de uma informação, a requisição de um serviço, uma advertência, etc. Considere as seguintes frases: "Sincronize o gerador 1." e "Seria possível sincronizar o gerador 1 ?". Ambas possuêm o mesmo conteúdo proposicional, mas a primeira frase expressa uma ordem e na segunda uma solicitação. Um tópico de pesquisa aberto em *IAD* é o quão extensiva deve ser a representação desses conceitos em primitivas ou no conteúdo da mensagem propriamente dita.

Uma outra idéia interessante que está sendo utilizada recentemente nesta área é o conceito de *sistemas governados por leis (do inglês "law-governed systems")* [MIN89]. A idéia central é que os protocolos para sistemas distribuídos devem ser "governados", para evitar sua violação. Ou seja toda a comunicação é governada por leis que possuem uma seqüência de regras predefinidas. Como por exemplo, as regras de trânsito podem ser consideradas conhecidas por todos. Quando um motorista aciona o pisca-pisca antes de dobrar à esquerda em um cruzamento, ele não está de fato se comunicando diretamente com os demais motoristas, mas estes conhecem sua intenção de dobrar a esquerda no próximo cruzamento, porque todos conhecem as leis de trânsito.

3.3.4 A Estrutura de controle de um Agente

As atividades que devem ser realizadas por um *agente* são:

- Percepção do ambiente onde ele está inserido.
- Comunicação com os demais *agentes da comunidade.*
- Planejamento estratégico para realização de tarefas.
- Executar atividades de solução de problemas.

Obviamente que uma estrutura complexa de controle precisa estar presente para assegurar um comportamento global coerente. Este controle pode ser *centralizado* ou *descentralizado.* O controle *centralizado* assemelha-se às organizações humanas altamente hierarquizadas, onde o agente detentor do conhecimento necessário à solução do problema irá dizer a todos os membros da *sociedade* o que fazer e quando. Por outro lado, no controle *descentralizado* os *agentes da comunidade* interagem para alocar as tarefas de acordo com as habilidades de cada um.

Assim como não há conceito sobre a definição do termo *agente*, não existe uma estrutura de controle universalmente aceita. Ou melhor, existem duas dimensões diferentes de controle: controle do agente e controle da sociedade. O controle do agente orienta como um *agente* deve organizar internamente suas atividades. Por outro lado o controle da sociedade orienta como organizar o conjunto de *agentes* e como controlar suas interações. Boisser [Boi92] tentou esclarecer estes conceitos, quando estava investigando como o *controle do agente* e o *da sociedade* poderiam ser divididos hierarquicamente, aplicando-se uma abordagem *SMA* aos problemas de visão computacional. Estes conceitos tiveram outra tentativa de formalização no trabalho de Levi [Lev90]. Do ponto de vista da Engenharia de "Software", Shoham [Sho92] propôs uma estrutura chamada "Agent-Oriented Programming", definindo os estados internos e os ciclos para um interpretador genérico de agentes. Ishida [Ish92] direcionou *controle social* e propôs um modelo para *organizações centralizadas para solução de problemas*, onde algumas estruturas de controle baseadas na realimentação entre as atividades de solução de problemas e organizações "auto-design" são definidas.

3.4 Linguagem para Comunicação de Agentes

A interação e a interoperação entre *agentes cognitivos*, para atingir objetivos comuns, requer mais que uma linguagem de comum entendimento dos *agentes* envolvidos nesse processo. O comportamento cooperativo em uma *comunidade de agentes* requer o cumprimento de três requisitos [FLM95]:

- **(1)** Uma linguagem em comum;
- **(2)** Um entendimento comum sobre a informação e o conhecimento compartilhados;

- **(3)** A habilidade para entender o que esta incluído em **(1)** e **(2)**.

Em se tratando de *comunidades de agentes cognitivos*, *(2)* e *(3)*, referem-se diretamente à base de conhecimento, e os mecanismos para comunicação de conhecimento presentes em cada agente. Neste caso um formalismo comum para representação de conhecimento é determinante para permitir que esses elementos sejam efetivos.

Os *agentes* são em sua maioria entidades computacionais residentes em um *Nível de Conhecimento* [FLM95] e eles não estariam bem servidos pelas linguagens e protocolos desenvolvidos para a computação distribuída. Estas linguagens e protocolos focalizam um *processo* ao invés de um programa ou uma coleção de programas que constituem um *agente*. Como resultado, uma linguagem de comunicação deve ser poderosa o suficiente para suportar a comunicação entre programas em um *alto nível*.

Uma linguagem de comunicação não é um protocolo. A distinção entre linguagem de comunicação e protocolo é difusa. Um protocolo, como os que são utilizados no contexto das linguagens de comunicação, pode apresentar um dos seguintes significados:

- Um protocolo de transporte como *ftp, http, etc.*
- Uma estrutura de interação de alto nível, *como negociação, protocolo da teoria de jogos*;
- Uma restrição às possíveis trocas válidas de primitivas de comunicação.

Uma linguagem de comunicação pode utilizar protocolos do primeiro tipo como mecanismo de transporte, pode ser usada por protocolos do segundo tipo como uma forma de implementação, e normalmente inclui protocolos do terceiro tipo. Mas definitivamente uma linguagem de comunicação não é meramente um protocolo [FLM95].

Uma Linguagem para Comunicação de Agentes *(LCA)* normalmente consiste em um conjunto de primitivas conhecido por todos os *agentes* da comunidade e um conjunto de regras de conversação. As *primitivas* informam o que esta sendo compartilhado e o que deve ser feito com este conhecimento ou informação. As regras de conversação, por sua vez, regulamentam as atitudes adotadas pelo agente durante a comunicação. Uma *LCA*, normalmente, é implementada em camadas, e como já foi dito, utilizam mecanismos para comunicação de processos provenientes de sistemas distribuídos. Finin, Labrou e Mayfield, sugeriram um conjunto de necessidades para uma *LCA* em *"KQML as an agent communication language"* [FLM95]. Estas necessidades são divididas em sete categorias: *forma, conteúdo, semântica, implementação, redes de comunicação, ambiente e confiabilidade.*

- **Forma** Uma boa *LCA* deve ser declarativa, sintaticamente simples e legível por seres humanos. Deve apresentar consistência, ser fácil de analisar e de gerar. Uma linguagem deve ser linear ou facilmente transformada em uma forma linear. Finalmente deve apresentar uma sintaxe extensível.

- **Conteúdo** Uma *linguagem para comunicação de agentes* deve ser organizada em camadas visando uma boa adaptação com outros sistemas. Em particular uma distinção deve ser feita entre linguagens de comunicação, que expressam atos comunicativos, e o conteúdo da linguagem, que expressa fatos sobre o domínio. A organização em camadas facilita a integração da linguagem com a aplicação, além de propiciar uma estrutura conceitual para o entendimento da linguagem. A linguagem deve ainda basear-se em um conjunto de atos comunicativos (primitivas).

- **Semântica** A semântica de uma linguagem de comunicação não deve ser ambígua, deve se possivel, apresentar, uma forma canônica, (isto é a similaridade de um significado deve levar à similaridade da representação).

- **Implementação** A implementação deve ser eficiente, tanto na velocidade quanto no espectro de utilização. Deve proporcionar uma boa integração com as tecnologias de "software" existentes. A interface deve ser de fácil utilização. Finalmente a linguagem deve ser amigável com implementações parciais.

- **Redes de Comunicação** Uma *linguagem para comunicação de agentes* deve adaptar-se bem às tecnologia modernas de redes de computadores. As linguagens devem suportar todos os tipos básicos de conexão: "ponto a ponto", "multicast" e "broadcast". As conexões síncronas e assíncronas devem ser suportadas. A linguagem deve ser formada por um conjunto de primitivas suficientemente rico para poder servir como um substrato onde linguagens de mais alto nível e protocolos de interação possam ser construídos. Mais ainda, estes protocolos de mais alto nível devem poder ser implementados independentemente do mecanismo de transporte utilizado.

- **Ambiente** O ambiente onde *os agentes inteligentes* serão requisitados para trabalhar será altamente distribuído, heterogêneo e extremamente dinâmico. Para propiciar um canal de comunicação neste ambiente, uma linguagem de comunicação precisa propiciar ferramentas para propiciar heterogeneidade e dinamismo. É necessário proporcionar a interoperação com outras linguagens e protocolos. Finalmente deve ser facilmente incorporada aos sistemas herdados.

- **Confiabilidade** Uma linguagem deve permitir uma comunição entre agente confiável e segura. Precauções para a segurança e conversações privadas entre dois agentes devem ser suportadas. A linguagem deve haver um forma de garantir autenticidade dos agentes. Deve ainda ser robusta o suficiente para suportar mensagens não apropriadas e mal formadas e deve ainda possuir mecanismos para identificação e sinalização de erros e advertências.

Dentre as *linguagens para comunicação de agentes*, as mais conhecidas são: *KQML "Knowledge Query Manipulation Language"*, que propõe um formato para as mensagens a serem trocadas, e um protocolo para manipulação das mensagens compartilhadas e para interação dos agentes; *CooL "Cooperação Language"*, uma linguagem baseada na *KQML*. Uma nova linguagem para comunicação de agentes, chamada *Parla* [CB97], utilizada no Expert-Coop [BC97] e será apresentada no Capitulo 4.

3.5 De Sistema Especialista Para Agente

Os Sistemas Especialistas, programas de computador concebidos para reproduzir o comportamento de especialistas humanos na solução de problemas do mundo real, representam uma concepção centralizada de comportamento inteligente. Por outro lado os *Sistemas Multi-Agentes* trabalham a questão da comunidade inteligente. Em um *SMA* o comportamento inteligente emerge da *sociedade de agentes*, podendo estes *agentes* individualmente possuir um comportamento inteligente ou não. Para que um sistema especialista possa tomar parte de uma *sociedade de agentes*, faz-se necessário adicionar algumas funcionalidades [SDB92]. Basicamente as funcionalidades a serem incorporadas são as seguintes:

- **Capacidade Perceptiva:** Um *agente* deve ser capaz de perceber as transformações do ambiente onde ele está inserido. Estas mudanças não estão apenas relacionadas com a interface homem-máquina, como nos *sistemas especialistas.* O ambiente pode sofrer modificações em conseqüência das ações de outro agentes;

- **Capacidade de Comunicação:** Um *agente* deve estar habilitado a comunicar-se com os demais *agentes da comunidade.* Isto não é comum nos sistemas especialistas. Mais que trocar dados, o termo comunicação empregado aqui envolve a troca de fragmentos de conhecimento e planejamentos parciais;

- **Capacidade de Atuar:** Um *agente* deve ser capaz de atuar em seu ambiente, como o resultado das suas atividades de solução de problemas. Mais uma vez, esta funcionalidade tem um sentido mais amplo que uma interface homem-máquina.

- **Comportamento Social:** Um *agente* deve ser capaz de raciocinar sobre as atividades dos demais *agentes.* Isto é realizado através de uma representação interna dos outros membros da comunidades e suas respectivas habilidades.

- **Estrutura de controle multi-camadas:** Um *agente* precisa decidir o momento mais apropriado para realização de suas atividades: percepção, comunicação e ação. Uma complexa estrutura de controle deve possibilitar a o acionamento destas atividades.

Capítulo 4

Expert-Coop

4.1 Introdução

A proposta inicial era dotar o *FASE* (ver capítulo 1) das funcionalidades necessária para que ele pudesse integrar uma sociedade de *agentes cognitivos* (ver capítulo 3). Pensou-se então em adicionar ao *FASE* mais uma biblioteca de funções dedicadas a realizar as atividades de comunicação com os *agentes da comunidade* onde o novo agente, gerado a partir do *FASE* seria inserido. Essa nova biblioteca seria composta por uma *linguagem de comunicação de agentes*, o suporte de comunicação, e um protocolo de cooperação. Essas três funcionalidades seriam proporcionadas pelo AgenTalk [KIO96]. Entretanto uma primeira objeção ocorreu, pois o AgenTalk só é compatível com o Allegro Common Lisp, o que exigiria um investimento de recursos não disponíveis no momento.

A impossibilidade de adquirir as licenças para o Allegro Common Lisp, levou-nos a buscar na Internet um Common Lisp compatível. Não foi encontrado nenhum Common Lisp que apresentasse a compatibilidade requerida pelo AgenTalk. Entretanto essa busca levou ao *CMU Common Lisp* [MaC92], um poderoso Common Lisp de domínio publico desenvolvido pela Carniege Mellon University, que inclui em seu pacote de distribuição o PCL, uma implementação do CLOS (extensão do Common Lisp à abordagem de Orientação a Objetos), um pacote para utilização de interface baseada no *Motif*, e principalmente dentre outros pacotes, um pacote chamado "wire", que permite a comunicação entre processos Lisp. Esse pacote "wire", baseado nos "sockets" do UNIX, permite que processos Lisp se comuniquem tanto no domínio do UNIX, possibilitando o desenvolvimento de sistemas concorrentes, como no domínio "INET", que permite o desenvolvimento de sistemas Lisp distribuídos. Essa comunicação se realiza segundo o modelo ISO/OSI. Uma vez encontrado um Common Lisp de domínio publico, que proporcionava um mecanismo para comunicação de processos Lisp em alto nível, resolveu-se implementar a *linguagem de comunicação de agentes* e protocolo de cooperação. Essa implementação deu origem mais tarde ao **Expert-Coop**.

O Expert-Coop foi concebido para facilitar a implementação de Sistemas Multi-Agentes Cognitivos, abertos e democráticos baseados em uma arquitetura pré-determinada. O ambiente possibilita tanto o desenvolvimento de uma *sociedade de agentes*, permitindo visualizar as interações passo a passo, quanto o desenvolvimento de um *agente cognitivo* para posteriormente ser inserido em uma *comunidade de agentes.* O Expert-Coop oferece um modelo de agente, uma linguagem de comunicado de agentes e um protocolo de cooperação, restando ao projetista a codificação do conhecimento, no Arcabouço de Sistemas Especialista *(o FASE)* encapsulado pelo processo *Expert.* Este ambiente para desenvolvimento de *SMA Cognitivos* é apresentado neste capítulo, abordando-se os seguintes aspectos: Arquitetura do Agente, Estratégia de Cooperação e Linguagem para Comunicação de Agentes.

4.2 Arquitetura do Agente

A arquitetura do agente suportado pelo Expert-Coop, mostrada na figura 4.1, é composta por seis processos concorrentes: Mail-box, Expert-box, Coordenador, Expert, dois processos mais que atualmente compõem a Interface e de um suporte de comunicação de alto nível. Estes processos interagem através de troca de mensagens.

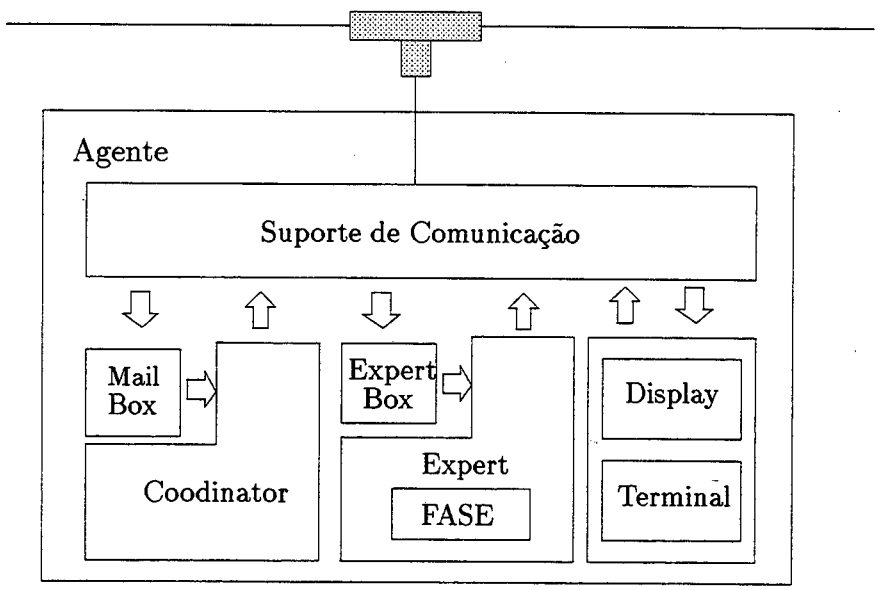

Figura 4.1: Arquitetura do Agente suportado pelo Expert-Coop

4.2.1 Mail-box

O processo Mail-box é dedicado a receber todas as mensagens, provenientes dos demais agentes da comunidades e dos processos Expert e Interface, e armazena tais mensagens em um "buffer". Este processo dispõe de dois *"wire's"*. O primeiro deles é utilizado para receber tanto as mensagens internas provenientes dos processos *Expert* e *Interface*, como as mensagens externas enviadas pelos demais *agentes* da comunidade. O segundo "wire" é utilizado exclusivamente pelo processo *Coordinator*, para ler as mensagens depositadas na "caixa postal" através de uma execução de procedimento remoto (*RPC, do inglês "Remote Procedure Call"*). Através dessa *RPC* as mensagens são transferidas para o processo *Coordinator* e a "caixa postal" esvaziada.

4.2.2 Coordinator

O processo Coordinator é responsável pelo comportamento cooperativo do *agente*. Todo o funcionamento deste processo está relacionado com as mensagens recebidas, de outros *agentes* ou processos. Cada ciclo de funcionamento deste processo é composto pelas seguintes atividades: leitura das mensagens, tratamento das mensagens, gerenciamento das licitações.

- **Leitura das Mensagens:** O ciclo de funcionamento do *Coordinator* inicia com a leitura das mensagens recebidas pelo *agente*. Essa leitura é realizada através de uma chamada de procedimento remoto *RPC* que transfere as mensagens recebidas pelo *Mail-Box* para o *Coordinator*. Uma vez transferidas, as mensagens serão desempacotadas, validadas, ordenadas segundo o algoritmo de "Ordem Total" [Lam78] e depositadas em uma lista.

- **Tratamento das Mensagens:** O tratamento das mensagens corresponde às ações que devem ser associadas a cada uma das mensagens recebidas: consulta ou informação destinada ao *Expert*, abertura de licitações requisitadas pelo processo *Expert*, apresentação de uma proposta a uma das licitações em curso, atualização dos *agentes ativos*, subscrição de um novo *agente* ou desligamento de um dos *agentes* da comunidade.

- **Gerenciamento das Licitações:** O gerenciamento das licitações corresponde ao fechamento das licitações, proclamação do vencedor e dos perdedores.

O processo *Coordinator* possui dois "wire's" destinados às atividades de comunicação, entretanto com propósitos distintos aos do Mail-box. O primeiro "wire" desse processo é utilizado para executar uma *RPC* no processo Mail-Box. O segundo "wire" é utilizado pelo processo *Coordinator* para enviar mensagens para os demais *agentes da comunidade* e para os processos *Export e Interface.*

4.2.3 Expert-box

O processo *Expert-box*, similar ao *Mail-box*, tem como objetivo receber todas as mensagens destinadas ao processo *Expert*. Essas mensagens são enviadas exclusivamente pelo *Coordinator*. O *Expert-box* foi adotado visando aumentar a concorrência entre os processos *Coordinator e Expert*, aumentando assim a eficiência do *agente*.

4.2.4 Expert

O processo Expert é responsável pelas habilidades cognitivas do agente. Este processo contém um sistema especialista encapsulado, mais todo o suporte de comunicação necessário para integrá-lo na arquitetura do agente. O sistema especialista associado ao processo Expert é baseado no FASE (ver secção 1.6).

Um manipulador de mensagens, chamado *Fase-Comm*, e uma nova estrutura para representação de conhecimento foram adicionados ao FASE. O *Fase-Comm* permite ao sistema especialista receber mensagens provenientes do processo Coordinator e responde-las como se estivesse manipulando conhecimento em sua base local. Tais características permitem incluir comandos para envio e recebimento de mensagens nas regras do sistema especialista. O funcionamento do processo *Expert* também é baseado em ciclos que são iniciados com a chegada de uma ou mais mensagem no *Expert-Box*. Estas mensagens representam um novo fato ou um questionamento. O ciclo de funcionamento do processo *Expert* é composto por três etapas: leitura das mensagens, inferência, comunicação.

- **Leitura das mensagens:** De forma similar ao processo *Coordinator*, as mensagens recebidas pelo *Expert-Box* são transferidas para o processo *Expert* através de uma *RPC*. Essa mensagens são desempacotadas, validadas, ordenadas, e o conhecimento contido nestas mensagens é armazenado na base de conhecimento do *FASE*.

- **Inferência:** Uma vez que novos fatos foram armazenados da base de conhecimento, são realizados os ciclos de inferência necessários. Casos estas inferências resultem em um conhecimento a ser exportado, este conhecimento é armazenado em uma estrutura apropriada, chamada de *export*.

- **Exportação:** Caso exista algum conhecimento a ser exportado, uma resposta a uma consulta, uma proposta a uma licitação aberta por um outro agente, uma requisição de informação externa, uma solicitação de abertura de licitação, etc, essa informação é enviada ao *Coordinator*.

4.2.5 Interface

A Interface, permite a interação entre o agente e o usuário. Atualmente esta interface é composta por dois processos: um terminal para enviar mensagens ao processo Coordinator e um "display" onde são mostradas as mensagens válidas, recebidas pelo agente. Encontra-se em desenvolvimento uma interface padrão utilizando as bibliotecas do *motif* que permitirá visualizar além das mensagens recebidas o histórico das licitações abertas pelo *agente* e seus respectivos estados. É possível ainda utilizar interfaces customizadas, ou seja direcionadas à aplicação em questão, baseadas nas bibliotecas do *Motif*.

4.3 Estratégia de Cooperação

O comportamento cooperativo da comunidade de agentes é obtido utilizando-se a estratégia de licitação. Quando um dos agentes da comunidade necessita realizar uma atividade cooperativa, ele abre uma licitação e difunde na comunidade o respectivo edital. Esta licitação tem um identificador único, que corresponde ao "time stamp" da mensagem que requisitou a licitação. Este mesmo identificador será utilizado no edital e nas propostas referentes à licitação em questão. Uma vez recebido um edital, os demais agentes da comunidade responderão com uma proposta, aceitando ou rejeitando a realização da tarefa licitada. Uma vez recebida uma proposta de cada um dos agentes ativos da comunidade, a melhor proposta é proclamada vencedora. O *agente* que colocou a proposta vencedora recebe uma mensagem informando que ele venceu a licitação e assumirá a responsabilidade da sua realização. Os demais *agentes* recebem uma mensagem retirando o edital. O requerente da licitação também é informado sobre quem foi o vencedor da licitação.

As propostas colocada pelos *agentes* da comunidade a um processo licitatório possuem um atributo chamado "grade", que pode ser um número "fuzzy", expressando o quão bem aquela tarefa pode ser realizada por aquele *agente.* A proposta que apresentar o maior "grade" será declarada a vencedora da licitação.

As licitações possuem um corte alfa que estabelece, um limite mínimo para o "grade" da proposta vencedora. Caso todas as proposta possuam um "grade" menor que o corte alfa, a licitação é fechada e o edital é retirado de todos os *agentes da comunidade.*

Os processos licitatórios podem ser abertos paralelamente por qualquer um dos agentes da comunidade, todos os agentes possuem o mesmo nível hierárquico, sendo que cada um dos agentes é responsável pelas licitações por ele abertas. E um mesmo *agente* pode abrir e gerir várias licitações simultaneamente.

4.4 Linguagem para Comunicação de Agentes

Atualmente existem algumas *linguagens para comunicação de agentes* já desenvolvidas, como por exemplo:

- **KQML** *"Knowledge Query Manipulation Language"* uma linguagem que propõe um formato para mensagens e um protocolo para manipulação destas mensagens, permitindo o compartilhamento de conhecimento entre agentes de uma dada comunidade.

- **CooL** *Cooperation Language* baseada na *KQML*, que inclui um protocolo de coordenação.

Entretanto o *KQML* requer um *agente especial* para prover as seguintes funções: associação do endereço físico do *agente* ao endereço simbólico; serviços de comunicação; um registro das bases de dados e/ou serviços oferecidos pelos outros *agentes da comunidade*. Além disso uma estruturação da linguagem em três camadas é adotada nessa linguagem: conteúdo, mensagem e comunicação. A primeira camada, a do conteúdo, traz o conteúdo da mensagem, o conhecimento ou informação. A camada de comunicação codifica um conjunto de mensagens que descrevem os parâmetros de comunicação de baixo nível. A camada da mensagem determina o tipo de interação que o *agente* realiza.

Uma nova *linguagem para comunicação de agentes*, chamada **Parla**, foi desenvolvida para ser utilizada pelo Expert-Coop, propondo um formato de mensagem alternativo e uma nova organização da estrutura de camadas permitindo que as atribuições que seriam realizadas pelo *agente especial* na proposta da *KQML*, possam ser reduzidas e estes novos requisitos passem a ser atribuições do suporte de comunicação do *agente*, ou ainda utilizar uma ferramenta de suporte a sistemas distribuídos como *ISIS* [Bir93].

4.4.1 Padrão de Mensagem

O padrão de mensagem adotado, apresentado abaixo, consiste em uma estrutura linear que contém os seguintes atributos (do inglês "slots"): From, To, Time-Stamp, Body, Grade, Round e Alfa. Dentre estes slots, From, To, Time-Stamp, Round e Body, são obrigatórios. Grade e alfa são facultativos.

```
((From agent-1)( To agent-2)( Time-Stamp 13) (Round 13)
 ( Body (ANNOUCE ((logic (sincronizar-grupo-gerador 2))))))
```

- **From** contém o nome do agente, ou processo no caso das mensagens entre processos de um mesmo *agente*.

- **To** armazena o nome do agente ou processo a que se destina a mensagem.

- **Time-Stamp** contém o "time stamp" da mensagem [Lam78].

- **Round** é utilizado para associar a mensagem a um dado processo licitatório em andamento.
- **Body** contém a mensagem propriamente dita.
- **Grade** permite atribuir um valor numérico, por exemplo um número "fuzzy", a uma proposta.
- **Alfa** permite estabelecer um corte alfa [DP80], para determinar o vencedor de uma licitação.

4.4.2 Estrutura de camadas

A *linguagem para comunicação de agentes Parla* suporta todos os tipos de comunicacao,"broadcast", "multicast" e "ponto-a-ponto", e está estruturada em apenas duas camadas - mensagem e comunicação - como mostra a figura 4.2.

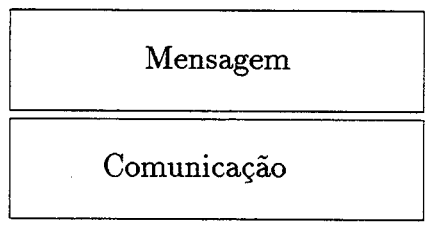

Figura 4.2: Estrutura de camadas da Linguagem Parla

- *Mensagem* - A camada da mensagem manipula os seguintes atributos da mensagem: *Body, To, Round, Grade* e *Alfa.* No atributos *Body* uma das primitivas da linguagem, seguida de um dos formalismos para representação de conhecimento suportados pelo *Expert*, lógica de primeira ordem, quadros e as redes semânticas. As primitivas informam ao *agente* o que deve ser feito com o conhecimento ou informação passado como parâmetro. O atributos *To* armazena o nome do *agente* ou grupo de *agentes* (all, loosers e etc.) a quem a mensagem é endereçada, *Round* refere-se ao identificador da licitação em questão, *Grade* é utilizado para atribuir um avaliador (coeficiente de certeza) a uma proposta, *Alfa* para estabelecer um corte Alfa para uma dada licitação.
- Comunicação - Esta camada manipula os atributos da mensagem referentes a algumas características da comunicação de baixo nível: *sender* o *agente* que enviou a mensagem e *time-stamp*, o "time stamp" da mensagem.

A Linguagem para Conunicação de Agentes, Parla, apresenta a seguinte sintaxe:

```
(<primitiva>  <formalismo de representação de conhecimento>)
```

4.4.3 Primitivas

As primitivas de comunicação que compõem a linguagem Parla, informam ao agente que tipo de imformação ou conhecimento contém a mensagem recebida e que ação deve ser tomada com a informação ou conhecimento em questão. As primitivas suportadas pela linguagem são: ANNOUNCE, ACCEPT, CONFIRM, INFORM, RECALL, REQUEST, REPLY, SUBSCRIBE e UNSUBSCRIBE.

- **ANNOUNCE:** Utilizada por um *agente*, para informar aos demais *agentes da comunidade* a abertura de uma licitação. Neste caso o objeto da licitação será representado no formalismo, passado como parâmetro, o atributos *Alfa* poderá ser utilizado para estabelecer um corte alfa para a licitação e o atributos *Round* obrigatoriamente conterá o identificador da licitação.

- **ACCEPT:** Utilizada por um ou mais *agentes da comunidade* para enviar uma proposta, aceitando realizar a tarefa licitada pelo *agente* que abriu a licitação. Essa proposta terá o mesmo *ROUND* da mensagem referente ao *ANNOUCE* e um quantificador numérico da proposta será armazenado no atributos *Grade*.

- **INFORM:** Utilizada para enviar uma informação a um processo ou *agente*.

- **CONFIRM:** Utilizada pelo agente responsável pela licitação para informar ao agente vencedor da licitação que sua proposta foi a vencedora. Essa proposta terá o mesmo *ROUND* da mensagem referente ao edital da licitação *(ANNOUCE)*.

- **RECALL:** Utilizada pelo agente responsável pela licitação para informar aos demais agentes que suas respectivas propostas não obtiveram êxito.

- **REFUSE:** Utilizada por um ou mais *agentes da comunidade* para enviar uma proposta, rejeitando realizar a tarefa licitada pelo *agente* que abriu a licitação. Essa proposta terá o mesmo *ROUND* da mensagem referente edital da licitação *(ANNOUCE)*.

- **REQUEST:** Utilizada para solicitar uma informação a um *agente* ou processo.

- **REPLY:** Utilizada para responder uma solicitação (REQUEST).

- **SUBSCRIBE:** Utilizada por um agente que não pertence à comunidade para solicitar sua inclusão. Esta solicitação pode ser feita a qualquer membro da comunidade e no formalismo de representação de conhecimento passado como parâmetro, deve aparecer o nome do agente e seu "host". Uma vez recebida tal solicitação o agente inclui o novo agente na lista de membros e difunde a nova lista de participantes da comunidade.

- **UNSUBSCRIBE:** Utilizado para solicitar a saída da comunidade.

O ambiente permite ainda adicionar facilmente novas primitivas à linguagem.

4.4.4 Regras de Comunicação

As primitivas de comunicação da linguagem *Parla* estão sujeitas as seguintes regras:

- A primitiva **ANNOUNCE** deve ser respondida com **ACCEPT** ou **REFUSE**.
- A primitiva **REQUEST** deve ser respondida com **REPLY**.
- As primitivas **SUBSCRIBE** e **UNSUBSCRIBE** devem invocar uma atualização dos *agentes ativos.*
- As primitivas **INFORM**, **CONFIRM** e **RECALL** são utilizadas para uma comunicação unidirecional.
- As licitações devem ser solicitadas pelo *Expert* com um **REQUEST** com o *Round* igual a zero. No fechamento da licitação requisitada um **REPLY** é enviado pelo *Coodinator* informando o vencedor.

Capítulo 5

Exemplo de Sistema Multi-Agentes Cognitivos

5.1 Introdução

A *Eletrosul*, empresa brasileira responsável pela geração e Transmissão de energia elétrica da região sul do Brasil, vem trabalhando em uma técnica para recomposição da rede de transmissão de energia elétrica chamada *Recomposição Fluente*. Neste processo cada uma das unidades da rede de transmissão de energia elétrica - usina hidroelétrica, termoelétrica ou subestação - possui seu próprio domínio de conhecimento sobre os equipamentos e procedimentos operacionais das respectivas unidades (ex. grupo gerador, linha de transmissão, disjuntores, barramentos etc.). Entretanto, informações globais tais como a completa topologia da rede, ou as intenções dos operadores de outras unidades não pertencem a esse domínio de conhecimento.

Uma vez ocorrido um "blackout" parcial ou total, cada um dos operadores das unidades da rede de transmissão, usinas hidroelétricas, termoeléctricas, subestações, etc, realiza um conjunto de ações predeterminadas em função das informações disponíveis em suas respectivas unidades, recompondo o maior número possível de unidades. Este conjunto de ações realizado pelos operadores das respectivas unidades, de acordo com o estado específico da unidade na rede é chamado de *Recomposição Fluente*. Esses procedimentos locais, passíveis de ser codificados em um sistema especialista, são coordenados por uma unidade central capaz de monitorar toda a rede, o *Centro de Operação do Sistema (COS)*. Por razões práticas esses procedimentos visam minimizar a comunicação entre os operadores das unidades em detrimento de uma operacionalidade ótima do sistema.

O problema na utilização desta estratégia é que a ocorrência de evento inesperado (ex. a perda de um grupo gerador ou de uma linha de transmissão, ou ainda o baixo nível

dos reservatórios) pode interromper o processo de recomposição fluente, necessitando de uma intervenção do *COS* para conduzir a continuação do processo de recomposição. Esta intervenção é realizada através de linhas de comunicação telefônicas privadas permitindo que os operadores das subestações e usinas se comuniquem com o *COS*. Essa comunicação é demorada e pode atrasar o processo de recomposição por um período considerado longo, podendo até exceder quinze minutos para cada intervenção do *COS*. Existem ainda situações onde a intervenção do *COS* é inevitável devido à centralização da informação.

Situações como estas podem ser manipuladas com mais eficiência, caso os operadores das plantas e subestações na vizinhança de onde o evento inesperado ocorreu, ou que possuam uma interdependência para realizar suas atividades, cooperassem entre si para solucionar o problema. Entretanto isto iria requerer uma comunicação intensa entre os operadores para que os requisitos de tempo real fossem cumpridos.

5.2 Abordagem Multi-Agentes

Uma abordagem Multi-Agentes é apresentada como proposta para solucionar alguns destes problemas que ocorrem durante o processo de *Recomposição Fluente*. A solução proposta consiste em alocar um agente cognitivo em cada subestação ou usina da rede onde será tratado o problema. Cada um destes *agentes* contém (na base de conhecimento do processo Expert) o conhecimento necessário sobre a respectiva unidade , subestação ou usina.

Dois tipos de formalismo para representação de conhecimento são utilizados em cada um dos *agentes*: Quadros e Lógica de primeira ordem. O formalismo dos quadros foi adotado para representar os elementos físicos da unidade (ex. grupo geradores, linhas de transmissão, disjuntores e etc.) e seus respectivos estados. Alguma variáveis de estado como a tensão, freqüência e a fase são representadas utilizando este formalismo. A Lógica de primeira ordem foi utilizada para representar o conhecimento necessário para permitir a cooperação entre os *agentes*. Em particular o método de tratamento de incerteza da Lógica Possibilística, disponível no *FASE* e por conseqüência no processo *Expert*, foi utilizado como quantificador, *Grade*, das propostas enviadas pelos *agentes* para atender a uma licitação aberta na comunidade.

A idéia central levantada pelo comportamento da *comunidade de agentes* proposta é que na ocorrência de uma situação anormal em alguma das unidades, caso uma determinada unidade necessite da ajuda de outras unidades da vizinhança, o respectivo *agente* pode iniciar um processo de cooperação entre os *agentes da comunidade*, que conduzirá para a solução do problema sem que haja a intervenção do *COS*. A identificação da situação e as respectivas trocas de mensagens necessárias à solução do problema podem ser codificadas em forma de regras no processo *Expert* existente em cada um dos *agentes da comunidade*.

5.3 Um Problema da Recomposição Fluente

Para ilustrar a utilização do Expert-Coop como ambiente para implementação de *sistemas multi-agentes cognitivos*, foi escolhido um dos casos da recomposição fluente onde a intervenção do *COS* é imprescindível. Uma parte da chamada Área 1, da rede de transmissão de energia elétrica é apresentada na figura 5.1. Três usinas hidroelétricas, *Salto Santiago (UHSS), Segredo (SGD) e Governador Bento Munhoz (GBM)*, e duas subestações, *Areia (ARE) e Ivaiporã (IVP)*, são apresentadas. Cada uma das usinas possui quatro grupo geradores, de diferentes potências: 380MW para cada um dos grupo geradores de UHSS, 360 MW para cada um dos grupo geradores de SGD, 425 MW para cada um dos grupo geradores de (GBM). Segundo a Instrução de Operação para a *Recomposição Fluente* [OPE95], para recompor a linha de transmissão que envia energia elétrica de *UHSS* para *IVP*, o operador da usina *UHSS* precisa certificar-se que existem ao menos *quatro dos doze grupo geradores* sincronizados, que a linhas de transmissão *LT-SGD*, *LT-ITA*, *LT-ARE* e *LT-ARE-1* já tenham sido recompostas e finalmente e que o "anel de sincronismo" entre *LT-ARE* e *LT-ARE-1* tenha sido fechado.

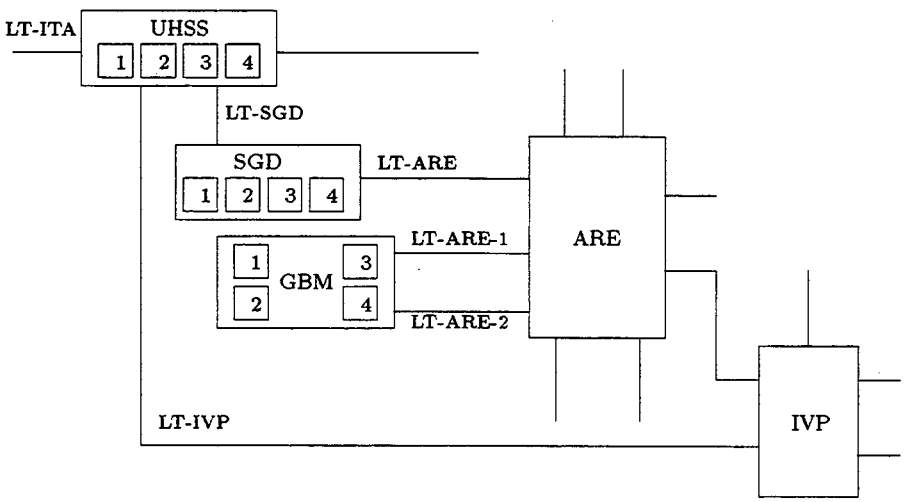

Figura 5.1: Parte da Rede de Transmissão de enegia elétrica da área 1 da Região Sul do Brasil

Solução: A solução consiste em alocar um *agente cognitivo* em cada uma das unidades envolvidas no problema. Devido ao fato de estarem disponíveis de apenas três estações de trabalho *(VEGA, SOL, POLARIS)* onde os *agentes do Expert-Coop* poderiam ser executados apresentando um rendimento aceitável, foram adotadas as seguintes hipóteses:

- A requisição para recompor a linha de transmissão que leva energia elétrica de *UHSS* para *IVP* parte da interface do agentes UHSS ao invés de ser enviada por um outro *agente* que estaria na subestação *IVP*.
- O anel de sincronismo entre *LT-ARE* e *LT-ARE-1* encontra-se fechado.

- As transformações do ambiente, ou seja a alteração dos estados dos geradores, barramentos disjuntores, etc. serão percebidos através de mensagens enviadas pelo terminal da interface.

Uma vez recebida a requisição para recompor a linha de transmissão, o *agente UHSS*, solicita aos *agentes SGD e GBM* o número de grupo geradores sincronizados em suas respectivas unidades. Caso o número de geradores sincronizados na comunidade não seja suficiente, ou seja inferior a quatro, o *agente UHSS* abrirá um determinado número de licitações visando completar o número de geradores necessários para recompor a linha de transmissão *LT-IVP*. Quando a licitação em questão receber uma proposta de cada um dos *agentes* ativos, esta licitação é fechada. Existindo um vencedor, este receberá uma mensagem informando-lhe que foi o vencedor da licitação, assumindo assim a responsabilidade pela sua realização. Os demais *agentes* serão informados que não obtiveram êxito no processo licitatório e a requisição para o respectiva licitação será respondida. Caso não haja um vencedor, todos receberam a informação que não obtiveram êxito na licitação. Completado o número de grupo geradores sincronizados na comunidade a linha de transmissão *LT-IVP* é recomposta e sua solicitação é respondida.

Nesta abordagem todos os *agentes* possuem o mesmo nível hierárquico e podem abrir processos licitatórios para satisfazer necessidades. A estratégia utilizada para realização de sucessivos processos licitatórios visando realizar a solicitação, foi baseada no número de grupo geradores que se deseja licitar:

- **Um grupo gerador:** Uma licitação para um gerador é aberta. E caso não haja vencedores a solicitação é respondida informando a impossibilidade de realizar a tarefa.

- **Mais que um grupo gerador:** Primeiramente tenta-se licitar diretamente dois a dois geradores. Caso não se obtenha êxito os geradores são licitado um a um.

A seguir são apresentados os conjuntos de mensagens trocadas entre os *agentes* da comunidade para três situações diferentes:

- **Situação 1:** Existem três grupos geradores sincronizados, um em cada usina hidroelétrica.

- **Situação 2:** Existem dois grupos geradores sincronizadas, um em *GBM* e outro em *SGD*. Um corte alfa no valor de 0.7 é especificado. E a usina *GBM* encontra-se com dois grupo geradores em manutenção.

- **Situação 3:** Nenhum dos geradores encontra-se sincronizado e foi estabelecido um corte alfa de 0.6.

Observe que:

- **Situação 1:** Necessita-se licitar um único grupo gerador, tabela 5.1. Durante o processo licitatório, todos os participantes, *GBM, UHSS, SGD*, depositam suas propostas aceitando realizar aquela tarefa, entretanto os *Grade's* das propostas são diferentes, vence então *GBM* que apresentou a proposta com maior *Grade*.

- **situação 2:** Inicialmente tenta-se licitar dois grupos geradores, tabelas 5.2 e 5.3. Neste primeiro processo licitatório o *agente GBM* apresenta um proposta recusando executar a tarefa, e os outros dois apresentam propostas aceitanto realizar a tarefa, entretanto, o *Grade* é inferior ao corte alfa estabelecido para a licitação. Como não houve vencedor para a licitação dos dois geradores necessários, eles são então licitados um a um.

- **situação 3:** Realiza-se primeiramente, uma licitação de dois grupos geradores onde a subestação *GBM* apresenta a melhor proposta vencendo assim a licitação, tabelas 5.4 e 5.5. Como ainda são necessários dois grupos geradores para completar o número mínimo de geradores sincronizados, é aberta uma nova licitação também com dois grupo geradores. Nesta segunda licitação todos os *agentes* da comunidade apresentam propostas aceitando a realização da tarefa, entretanto nenhuma das proposta possui um *Grade* superior ao corte alfa estabelecido para a licitação. Então estes dois geradores restantes são licitados um a um.

Message Body	From	To	Rd	Gd	T.S.
(REQUEST ((logic (recompose lt uhss-ivp))))	Inter.	Coord.			1
(REQUEST ((logic (recompose lt uhss-ivp))))	Coord.	Expert			3
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			5
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			8
(REPLY ((logic (synchronized-generators sgd 1))))	SGD	UHSS			12
(REPLY ((logic (synchronized-generators gbm 1))))	GBM	UHSS			15
(REPLY ((logic (synchronized-generators sgd 1))))	Coord.	Expert			18
(REPLY ((logic (synchronized-generators gbm 1))))	Coord.	Expert			21
(REQUEST ((logic (synchronize-generator x 1))))	Expert	Coord.	0		23
(ANNOUCE ((logic (synchronize-generator x 1))))	UHSS	All	26		28
(ANNOUCE ((logic (synchronize-generator x 1))))	Coord.	Expert	26		32
(ACCEPT ((logic (synchronize-generator gbm 1))))	GBM	UHSS	26	0.8	34
(ACCEPT ((logic (synchronize-generator sgd 1))))	SGD	UHSS	26	0.6	36
(ACCEPT ((logic (synchronize-generator uhss 1))))	Expert	Coord.	26	0.6	41
(CONFIRM ((logic (synchronize-generator gbm 1))))	UHSS	GBM	26	0.8	47
(RECALL ((logic (synchronize-generator x 1))))	LOOSERS	SGD	26	0.6	49
(RECALL ((logic (synchronize-generator x 1))))	Coord.	Expert	26	0.6	52
(REPLY ((logic (synchronize-generator gbm 1))))	Coord	Expert		0.8	54
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			56
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			60
(REPLY ((logic (synchronized-generators sgd 1))))	SGD	UHSS			62
(REPLY ((logic (synchronized-generators gbm 1))))	GBM	UHSS			65
(REPLY ((logic (synchronized-generators sgd 1))))	Coord.	Expert			68
(REPLY ((logic (recomposed lt uhss-ivp))))	Expert	Coord.			70

Tabela 5.1: Situação 1

Message Body	From	To	Rd	Gd	T.S.
(REQUEST ((logic (recompose lt uhss-ivp))))	Inter.	Coord.			1
(REQUEST ((logic (recompose lt uhss-ivp))))	Coord.	Expert			3
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			4
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			8
(REPLY ((logic (synchronized-generators sgd 1))))	SGD	UHSS			12
(REPLY ((logic (synchronized-generators gbm 1))))	GBM	UHSS			15
(REPLY ((logic (synchronized-generators sgd 1))))	Coord.	Expert			18
(REPLY ((logic (synchronized-generators gbm 1))))	Coord.	Expert			21
(REQUEST ((logic (synchronize-generator x 2))))	Expert	Coord.	0		23
(ANNOUCE ((logic (synchronize-generator x 2))))	UHSS	All	26		28
(ANNOUCE ((logic (synchronize-generator x 2))))	Coord.	Expert	26		32
(REFUSE ((logic (synchronize-generator gbm 2))))	GBM	UHSS	26	0.8	34
(REFUSE ((logic (synchronize-generator sgd 2))))	SGD	UHSS	26	0.6	36
(ACCEPT ((logic (synchronize-generator uhss 1))))	Expert	Coord.	26	0.6	41
(RECALL ((logic (synchronize-generator x 1))))	UHSS	ALL	26	0.6	47
(RECALL ((logic (synchronize-generator x 1))))	Coord.	Expert	26	0.6	49
(REPLY ((logic (synchronize-generator NO-WINNER 2))))	Coord	Expert		0.8	52
(REQUEST ((logic (synchronize-generator x 1))))	Expert	Coord.	0		54
(ANNOUCE ((logic (synchronize-generator x 1))))	UHSS	All	56		58
(ANNOUCE ((logic (synchronize-generator x 1))))	Coord.	Expert	56		60
(ACCEPT ((logic (synchronize-generator gbm 1))))	GBM	UHSS	56	0.6	63
(ACCEPT ((logic (synchronize-generator sgd 1))))	SGD	UHSS	56	0.6	65
(ACCEPT ((logic (synchronize-generator uhss 1))))	Expert	Coord.	56	0.8	68
(CONFIRM ((logic (synchronize-generator uhss 1))))	Cood.	Expert	56	0.8	70
(RECALL ((logic (synchronize-generator x 1))))	UHSS	LOOSERS	56	0.6	74
(REPLY ((logic (synchronize-generator uhss 1))))	Coord	Expert		0.8	76

Tabela 5.2: Situação 2

Message Body	From	To	Rd	Gd	T.S.
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			78
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			82
(REPLY ((logic (synchronized-generators sgd 1))))	SGD	UHSS			85
(REPLY ((logic (synchronized-generators gbm 1))))	GBM	UHSS			88
(REPLY ((logic (synchronized-generators sgd 1))))	Coord.	Expert			90
(REQUEST ((logic (synchronize-generator x 1))))	Expert	Coord.	0		92
(ANNOUCE ((logic (synchronize-generator x 1))))	UHSS	All	94		96
(ANNOUCE ((logic (synchronize-generator x 1))))	Coord.	Expert	94		98
(ACCEPT ((logic (synchronize-generator gbm 1))))	GBM	UHSS	94	0.6	101
(ACCEPT ((logic (synchronize-generator sgd 1))))	SGD	UHSS	94	0.6	104
(ACCEPT ((logic (synchronize-generator uhss 1))))	Expert	Coord.	94	0.8	106
(CONFIRM ((logic (synchronize-generator uhss 1))))	Cood.	Expert	94		108
(RECALL ((logic (synchronize-generator x 1))))	UHSS	LOOSERS	94		111
(REPLY ((logic (synchronize-generator uhss 1))))	Coord	Expert		0.8	113
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			115
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			119
(REPLY ((logic (synchronized-generators sgd 1))))	SGD	UHSS			122
(REPLY ((logic (synchronized-generators gbm 1))))	GBM	UHSS			125
(REPLY ((logic (synchronized-generators sgd 1))))	Coord.	Expert			127
(REPLY ((logic (recomposed lt uhss-ivp))))	Expert	Coord.			129

Tabela 5.3: Situação 2 (continuação)

Message Body	From	To	Rd	Gd	T.S.
(REQUEST ((logic (recompose lt uhss-ivp))))	Inter.	Coord.			1
(REQUEST ((logic (recompose lt uhss-ivp))))	Coord.	Expert			3
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			4
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			8
(REPLY ((logic (synchronized-generators sgd 0))))	SGD	UHSS			12
(REPLY ((logic (synchronized-generators gbm 0))))	GBM	UHSS			15
(REPLY ((logic (synchronized-generators sgd 0))))	Coord.	Expert			18
(REPLY ((logic (synchronized-generators gbm 0))))	Coord.	Expert			21
(REQUEST ((logic (synchronize-generator x 2))))	Expert	Coord.	0		23
(ANNOUCE ((logic (synchronize-generator x 2))))	UHSS	All	26		28
(ANNOUCE ((logic (synchronize-generator x 2))))	Coord.	Expert	26		33
(ACCEPT ((logic (synchronize-generator gbm 2))))	GBM	UHSS	26	1.0	35
(ACCEPT ((logic (synchronize-generator sgd 2))))	SGD	UHSS	26	0.7	37
(ACCEPT ((logic (synchronize-generator uhss 2))))	Expert	Coord.	26	0.8	42
(CONFIRM ((logic (synchronize-generator gbm 2))))	UHSS	GBM	26	0.8	44
(RECALL ((logic (synchronize-generator x 2))))	UHSS	LOOSERS	26	0.6	48
(RECALL ((logic (synchronize-generator x 2))))	Coord.	Expert	26	0.6	50
(REPLY ((logic (synchronize-generator gbm 2))))	Coord	Expert		0.8	52
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			54
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			59
(REPLY ((logic (synchronized-generators sgd 0))))	SGD	UHSS			62
(REPLY ((logic (synchronized-generators gbm 2))))	GBM	UHSS			65
(REPLY ((logic (synchronized-generators sgd 0))))	Coord.	Expert			67
(REPLY ((logic (synchronized-generators gbm 2))))	Coord.	Expert			69
(REQUEST ((logic (synchronize-generator x 2))))	Expert	Coord.	0		71
(ANNOUCE ((logic (synchronize-generator x 2))))	UHSS	All	72		74
(ANNOUCE ((logic (synchronize-generator x 2))))	Coord.	Expert	72		76
(ACCEPT ((logic (synchronize-generator gbm 2))))	GBM	UHSS	72	0.5	79
(ACCEPT ((logic (synchronize-generator sgd 2))))	SGD	UHSS	72	0.6	82
(ACCEPT ((logic (synchronize-generator uhss 2))))	Expert	Coord.	72	0.6	84
(RECALL ((logic (synchronize-generator x 2))))	ALL	ALL	72	0.6	89
(RECALL ((logic (synchronize-generator x 2))))	Coord.	Expert	72	0.6	91
(REPLY ((logic (synchronize-generator NO-WINNER 2))))	Coord	Expert		0.8	93

Tabela 5.4: Situação 3

Message Body	From	To	Rd	Gd	T.S.
(REQUEST ((logic (synchronize-generator x 1))))	Expert	Coord.	0		95
(ANNOUCE ((logic (synchronize-generator x 1))))	UHSS	All	96		98
(ANNOUCE ((logic (synchronize-generator x 1))))	Coord.	Expert	96		100
(ACCEPT ((logic (synchronize-generator gbm 1))))	GBM	UHSS	96	0.6	103
(ACCEPT ((logic (synchronize-generator sgd 1))))	SGD	UHSS	96	0.6	106
(ACCEPT ((logic (synchronize-generator uhss 1))))	Expert	Coord.	96	0.8	108
(CONFIRM ((logic (synchronize-generator uhss 1))))	Cood.	Expert	96	0.8	110
(RECALL ((logic (synchronize-generator x 1))))	UHSS	LOOSERS	96	0.6	114
(REPLY ((logic (synchronize-generator uhss 1))))	Coord	Expert		0.8	116
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			118
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			123
(REPLY ((logic (synchronized-generators sgd 0))))	SGD	UHSS			126
(REPLY ((logic (synchronized-generators gbm 2))))	GBM	UHSS			129
(REPLY ((logic (synchronized-generators sgd 0))))	Coord.	Expert			131
(REPLY ((logic (synchronized-generators gbm 2))))	Coord.	Expert			133
(REQUEST ((logic (synchronize-generator x 1))))	Expert	Coord.	0		135
(ANNOUCE ((logic (synchronize-generator x 1))))	UHSS	All	137		139
(ANNOUCE ((logic (synchronize-generator x 1))))	Coord.	Expert	137		141
(ACCEPT ((logic (synchronize-generator gbm 1))))	GBM	UHSS	137	0.6	143
(ACCEPT ((logic (synchronize-generator sgd 1))))	SGD	UHSS	137	0.6	147
(ACCEPT ((logic (synchronize-generator uhss 1))))	Expert	Coord.	137	0.8	149
(CONFIRM ((logic (synchronize-generator uhss 1))))	Cood.	Expert	137	0.8	151
(RECALL ((logic (synchronize-generator x 1))))	UHSS	LOOSERS	137	0.6	155
(REPLY ((logic (synchronize-generator uhss 1))))	Coord	Expert		0.8	157
(REQUEST ((logic (synchronized-generators x y))))	Expert	Coord.			159
(REQUEST ((logic (synchronized-generators x y))))	UHSS	All			163
(REPLY ((logic (synchronized-generators sgd 1))))	SGD	UHSS			166
(REPLY ((logic (synchronized-generators gbm 1))))	GBM	UHSS			169
(REPLY ((logic (synchronized-generators sgd 1))))	Coord.	Expert			172
(REPLY ((logic (recomposed lt uhss-ivp))))	Expert	Coord.			174

Tabela 5.5: Situação 3 (continuação)

Capítulo 6

Conclusões e Perspectivas

6.1 Conclusões

O *Expert-Coop* é uma ferramenta flexível, de domínio público, para a implementação de *Sistemas Multi-Agentes Cognitivos, Democráticos, Heterogêneos e Abertos.* Os *SMA's* construídos a partir do *Expert-Coop* não necessitam qualquer tipo de hierarquia entre os participantes da *comunidade de agentes.* A estratégia de cooperação utilizada pelos *agentes* é baseada na abertura de licitações para realização das tarefas cooperativas. As licitações podem ser abertas por qualquer um dos *membros da comunidade*, e cada um dos agentes é capaz de gerir vários processos licitatórios concorrentemente. A estrutura do ambiente foi definida utilizando-se o CLOS, uma extensão do Common Lisp à abordagem de Orientação a Objetos, que facilita tanto as modificações das estratégias de cooperação já implementadas como a inclusão de novas estratégias. Outras características importantes do ambiente são: a portabilidade, modularidade, a existência de mecanismos de herança e a possibilidade de utilização de interfaces gráficas baseadas no *Motif.*

Os *agentes cognitivos* construídos a partir do *Expert-Coop*, são baseados em um modelo genérico para *agentes cognitivos.* Esse modelo genérico apresenta uma arquitetura concorrente, baseada em seis processos: *Mail-Box, Coordinator, Expert-Box, Expert, e uma Interface composta por um processo Terminal e um processo Display.* Estes processos apresentam um alto grau de paralelismo, o que se reflete diretamente no desempenho do *agente.* A Capacidade cognitiva do *agente* é realizada pelo processo *Expert*, baseado no arcabouço para sistemas especialistas *FASE.* O sistema *FASE* suporta os três tipos de representação de conhecimento mais utilizados, lógica de primeira ordem, quadros, redes semânticas. Uma nova estrutura para representação de conhecimento chamada *export* foi adicionada à base de conhecimento. Essa nova estrutura para representação de conhecimento é utilizada pelo sistema especialista que compõe o processo *Expert* para armazenar o conhecimento a ser compartilhado com outros *agentes da comunidade.*

O protótipo de Sistemas Multi-Agentes, implementado a partir do Expert-Coop neste trabalho de dissertação, evidenciou a aplicabilidade desta tecnologia emergente em um problema do mundo real. Mostrou-se promissor a utilização de um Sistema Multi-Agentes para toda a malha de transmissão de energia elétrica. Uma abordagem Multi-Agentes aplicada a recomposição da rede de transmissão de energia elétrica, por exemplo, permititria atingir a melhor configuração para recompor o sistema em um curto periodo de tempo, qualquer que seja o estado da rede. Mostrou ainda ser possível a utilização o *Expert-Coop* na implementação de Sistemas Multi-Agentes, destinados a tratar problemas do mundo real caracterizados pela existencia de um conhecimento distribuido quer seja pela necessidade de envolver diversas fontes de conhecimento e/ou pela distribuição geográfica destas, como por exemplo, abastecimento e distribuição de fluídos, Controle de Tráfego Aéreo e hurbano, sistemas para triagem de pacientes, etc.

6.2 Perspectivas

O *Expert-Coop*, deve ser visto como um ponto de partida para o desenvolvimento de trabalhos de pesquisa ou implementações práticas na área de sistemas multi-agentes, utilizando-se o modelo de *agente cognitivo genérico* e a linguagem para comunicação de agentes, *Parla*, propostos. Dentre estes possíveis trabalhos podem ser citados os seguintes:

- A implementação de sistemas *multi-agentes cognitivos* com elevado grau de complexidade, como por exemplo nas áreas de: geração e transmissão de energia elétrica, sistemas de abastecimento de água, controle de tráfego urbano e aéreo, simulação distribuída de processos, integração de sensores de aeronaves, sistemas de triagem de pacientes de hospitais e clínicas.

- Incorporação de ferramentas para ambientes distribuídos, como por exemplo *ISIS Toolkit*, no *Expert-Coop.*

- Construção de um lisp.core para o *CMU Common Lisp* dedicado ao *Expert-Coop*, para reduzir a quantidade de memória necessária para utilização do ambiente.

- Inclusão de estratégias de cooperação para *sociedade de agentes* baseadas em negociação.

- Desenvolvimento de uma interface gráfica para utilização do *Expert-Coop.*

- Inclusão de estratégias de cooperação para *sociedade de agentes* baseadas em negociação.

- desenvolvimento de novas estratégias de cooperação e negociação para *sociedades de agentes cognitivos.*

Apêndice A

Agente UHSS

A.1 Construção do Agente UHSS

A construção do *agente cognitivo* a partir do Expert-Coop consiste em disparar seis processos lisp:

- mbox.lisp
- fbox.lisp
- display.lisp
- sender.lisp
- expert.lisp
- coord.lisp

Os três primeiros processos dispensam qualquer tipo de parâmetro, ou de configuração, podendo ser utilizado um "script" para cada um deles como é mostrado a seguir.

A.1.1 Processo Mail-box

O processo Mail-box dispensa qualquer tipo de configuração ou passagem de parâmetro. Para acioná-lo basta abrir uma seção lisp na maquina onde o *agente* será alocado e carregar o arquivo "mbox.lisp" (load "mbox"). Ou utilizar um "script" como o mostrado a seguir. Este "script", chamado de *mbox-1* ira alocar o processo *Mail-box* do *agente UHSS* na estação de trabalho "vega".

```
#!/bin/tcsh -f
rsh vega setenv DISPLAY $1 &
rsh vega /opt/cmulisp/bin/common-lisp -init $HOME/Agentes/src/mbox.lisp
```

A.1.2 Processo Expert-box

O processo Expert-box, assim como o Mail-box, também dispensa qualquer tipo de configuração ou passagem de parâmetro. Um "script" chamado *fbox* irá alocar o processo *Expert-box* do *agente* do *agente UHSS* na estação de trabalho "vega".

```
#!/bin/tcsh -f
rsh vega setenv DISPLAY $1 &
rsh vega /opt/cmulisp/bin/common-lisp -init $HOME/Agentes/src/fbox.lisp
```

A.1.3 Processo Display

Assim como *Expert-box e Mail-box* o processo display também dispensa a passagem de parâmetros ou qualquer tipo de configuração. Um "script" chamado display irá alocar o processo Display do *agente* UHSS na máquina "vega"

```
#!/bin/tcsh -f
rsh vega setenv DISPLAY $1 &
rsh vega /opt/cmulisp/bin/common-lisp -init $HOME/Agentes/src/display.lisp
```

A.1.4 Processo Terminal

O processo Terminal necessita que seja informado o nome da máquina para onde serão enviadas as mensagens. Esta informação é passada no arquivo "sender.lisp" no seguinte trecho.

```
(setq sender1 (make-instance 'sender :port-to 9000 :host-to "vega"))
```

Por razões práticas foram criados os arquivos "sender-1.lisp", "sender-2.lisp" e "sender-3.lisp", já configurados para as estações de trabalho "vega", "polaris" e "sol" respectivamente. Foram incluídos também nestes arquivos funções que disparam as mensagens que precisam ser enviadas no exemplo apresentado no capítulo cinco. Sendo assim temos o "script" sender-1 que implementa o *processo Terminal* já configurado para o agente *UHSS* na maquina "vega".

```
#!/bin/tcsh -f
rsh vega setenv DISPLAY $1 &
rsh vega /opt/cmulisp/bin/common-lisp -init $HOME/Agentes/src/sender-1.lisp
```

A.1.5 Processo Expert

Este processo precisa de três informações como parâmetros: o arquivo de fatos, o arquivo de regras, e o "host". Os arquivos de fatos e regras são definidos nos arquivos "kb-def1.lisp", "kb-def2.lisp" e "kb-def3.lisp". Também por motivos práticos foram criados os arquivos "Expert-1.lisp", "Expert-2.lisp", "Expert-3.lisp", que contem os respectivos arquivos de fatos e regras além do host para os *agentes UHSS, SGD, GBM.* Sendo assim o "scritp" "Expert-1" responsável pelo processo *Expert* do *agente UHSS.*

```
#!/bin/tcsh -f
rsh vega setenv DISPLAY $1 &
rsh vega /opt/cmulisp/bin/common-lisp -init $HOME/Agentes/src/Expert-1.lisp
```

A.1.6 Processo Coordinator

Este processo possui os seguintes parâmetros a configurar: lista dos agentes da comunidade, localização dos agentes ("host" e porta). Também por motivos práticos foi adotado um arquivo de configuração para cada um dos processos coordinator's "uhss.lisp", "sgd.lisp" e "gbm.lisp". O "script" Coord-1 é responsavel pelo processo *coodinator* do *agente UHSS.*

```
;; Arquivo de inicializacao do uhss.lisp
;;
;;

(load "/home/mestrado/loureiro/Agentes/src/coord.lisp")
(setq *agent* (make-instance 'agent
       :agent-list (list 'sgd 'gbm )
       :agent-msg-bsize 1
       :agent-box (wire:connect-to-remote-server "vega" 9001)
       :agent-name 'uhss
       :fase-sender (make-instance 'sender
                                   :port-to 7000
                                   :host-to "vega"
                                   :sender-id 'FASE)

       :display-sender (make-instance 'sender
                                   :port-to 4000 :host-to "vega"
                                   :sender-id 'DISPLAY)))
```

```
;;
;; Sender's setup
;;

(with-slots (agent-sender) *agent*
     (setf agent-sender (list
      (make-instance 'sender :port-to 9000
                             :host-to "polaris"
                             :sender-id 'sgd)

      (make-instance 'sender :port-to 9000
                             :host-to "sol"
                             :sender-id 'gbm))))
```

```
#!/bin/tcsh -f
rsh vega setenv DISPLAY $1 &
rsh vega /opt/cmulisp/bin/common-lisp -init $HOME/Agentes/src/uhss.lisp
```

A.1.7 O Agente

Um "script" que promove toda a construção do *agente* invocando os demais "script's" aqui descritos tambem está disponivel. Agent-1 para *o agente UHSS*, Agent-2 para *SGD* e Agent-3 para *GBM*.

```
#!/bin/tcsh -f
dtterm -iconic -title mbox-1 -name mbox-1 -e mbox-1 {$HOST}:0.0 &
dtterm -iconic -title fbox-1 -name fbox-1 -e fbox-1 {$HOST}:0.0 &
dtterm -title Expert-1 -name Expert-1 -geometry 100x13+70+242 -e Expert-1
{$HOST}:0.0 &
dtterm  -title display-1 -name display-1 -geometry 108x15+0+0 -e display-1
{$HOST}:0.0 &
dtterm  -title sender-1 -iconic -name sender-1 -e sender-1 {$HOST}:0.0 &
sleep 30
dtterm  -title Agent-1 -name Agent-1 -geometry 100x13+70+460 -e Coord-1
{$HOST}:0.0 &
```

Sendo assim para ativar o *agente UHSS* basta digitar "Agent-1" em um terminal do UNIX.

A.2 Arquivo de fatos

```
((frame (hidro root)
        (uhss hidro
                  (v-nom 525)
                  (f-nom 60)
                  (v-uhss 0)
                  (f-uhss 0)
                  (fase-uhss 0)
                  (site area-1)
                  (bus-A off)
                  (reserved-generators 0)
          (bus-B off))
         (tf-1 uhss
           (t-state  off))
             (generator-1 tf-1
                        (v 0)
                        (f 0)
                        (fase 0)
                        (state off))

             (generator-2 tf-1
                        (v 0)
                        (f 0)
                        (fase 0)
                        (state off))

         (tf-2 uhss
             (t-state  off))
               (generator-3 tf-2
                        (v 0)
                        (f 0)
                        (fase 0)
                        (state off))


              (generator-4 tf-2
                        (v 0)
                        (f 0)
                        (fase 0)
                        (state off))
```

```
        (LT uhss
                    (lt-ivp off)
                    (lt-sgd off)
                    (lt-ita off))

        (MW uhss
                    (MW-ivp 0)
                    (MW-sgd 0)
                    (MW-ita 0))

         (cb uhss
                     (cb1022 off)
                     (cb1030 off)
                     (cb1032 off)
                     (cb1062 off)
                     (cb1070 off)
                     (cb1082 off)
                     (cb1090 off)
                     (cb1092 off))
         (network uhss
                     (gbm  alive)
                     (sgd  alive)))
 (logic  (total-syncronized-generator unknowed)
         (update-generators off)))
```

A.3 Arquivo de regras

```
(regra-1 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-1 tf-1  (state x1))
                 (generator-2 tf-1  (state x2))
                 (generator-3 tf-2  (state x3))
                 (generator-4 tf-2  (state x4))
                 (uhss  hidro (bus-B off)))
          (lisp (and (equal 'x2 'off)
                     (equal 'x1 'off))))
         ((frame (generator-1 tf-1  (state on)))
          (logic (off (syncronize generator-group 1)))))
```

```
(regra-2 1.0
        ((logic (syncronize generator-group 1))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (uhss  hidro (bus-B off)))
         (lisp (or  (equal 'x2 'maintenence)
                    (equal 'x1 'maintenence)
                    (and (equal 'x1 'on) (equal 'x2 'on))
                    (and (equal 'x1 'syncronized )
                         (equal 'x2 'syncronized )))))
        ((frame (generator-1 tf-1  (state on)))
         (logic (off (syncronize generator-group 1)))))

(regra-3 1.0
        ((logic (syncronize generator-group 1))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (uhss  hidro (bus-B off)))
         (lisp (and (< 1 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (or (and (equal 'x1 'on)(equal 'x2 'on))
                        (or  (equal 'x1 'maintenance )
                             (equal 'x1 'maintenance ))))))
        ((frame (generator-3 tf-2  (state on)))
         (logic (off (syncronize generator-group 1)))))

(regra-4 1.0
        ((logic (syncronize generator-group 1))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (uhss  hidro (bus-B off)))
         (lisp (and (< 1 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'x1 'on)
                    (equal 'x2 'off))))
        ((frame (generator-2 tf-1  (state on)))
         (logic (off (syncronize generator-group 1)))))
```

```
(regra-5 1.0
        ((logic (syncronize generator-group 1))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (uhss  hidro (bus-B off)))
         (lisp (and (< 1 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'x2 'on)
                    (equal 'x1 'off))))
        ((frame (generator-1 tf-1  (state on)))
         (logic (off (syncronize generator-group 1)))))
```

```
(regra-6 1.0
        ((logic (syncronize generator-group 1))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (uhss  hidro (bus-B off)))
         (lisp (and (< 1 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'x3 'on)
                    (equal 'x4 'off))))
        ((frame (generator-4 tf-2  (state on)))
         (logic (off (syncronize generator-group 1)))))
```

```
(regra-7 1.0
        ((logic (syncronize generator-group 1))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (uhss  hidro (bus-B off)))
         (lisp (and (< 1 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'x4 'on)
                    (equal 'x3 'off))))
        ((frame (generator-3 tf-2  (state on)))
         (logic (off (syncronize generator-group 1)))))
```

```
(regra-8 1.0
        ((logic (syncronize generator-group y))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4))
                (y1 y2  (state off)))
         (lisp (eql 'y (n-times 'off (list 'x1 'x2 'x3 'x4 )))))
        ((logic (off (syncronize generator-group 2)))
         (frame (y1 y2  (state on)))))


(regra-9 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4)))
         (lisp (and (eql 4 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x1)
                    (equal 'off 'x2))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 tf-1  (state on))
                (generator-2 tf-1  (state on)))))


(regra-10 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4)))
                    (equal 'off 'x2)
                    (equal 'off 'x3)
                    (equal 'off 'x4))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-3 tf-1  (state on))
                (generator-4 tf-1  (state on)))))
```

```
(regra-11 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4)))
                    (equal 'off 'x1)
                    (equal 'off 'x2)
                    (equal 'off 'x3))))
         ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 tf-1  (state on))
                (generator-2 tf-1  (state on)))))


(regra-12 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4)))
                    (equal 'off 'x1)
                    (equal 'off 'x3)
                    (equal 'off 'x4))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-3 tf-2  (state on))
                (generator-4 tf-2  (state on)))))


(regra-13 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 tf-1  (state x1))
                (generator-2 tf-1  (state x2))
                (generator-3 tf-2  (state x3))
                (generator-4 tf-2  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4)))
                    (equal 'off 'x1)
                    (equal 'off 'x2)
                    (equal 'off 'x4))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 tf-1  (state on))
                (generator-2 tf-1  (state on)))))
```

```
(regra-14 1.0
        ((frame (uhss hidro (v-uhss x1)(f-nom x2)(v-nom x4))
                (x5 x6 (state on)(v x7)(f y1)(fase y2)))
         (lisp (and ( = 0 x1)
                    (> (* 0.1 x4) (abs (-  x7 x4)))
                    (> 0.2 (abs (- x2 y1))))))
        ((frame (x5 x6 (state syncronized))
                (x6 uhss (t-state on))
                (uhss hidro (v-uhss x7)(f-uhss y1)(fase-uhss y2)))))


(regra-15 1.0
        ((frame (uhss hidro (v-uhss x1)(f-uhss x2)
                            (fase-uhss x4)(v-nom y1))
                (x5 x6 (state on)(v x7)(f y2)(fase y3)))
         (lisp (and (> x1 (* 0.9 y1))
                    (> (* 0.1 y1) (abs (-  x7 x1)))
                    (> 0.2 (abs (- x2 y2)))
                    (> 10 (abs (-  x4 y3))))))
        ((frame (x5 x6 (state syncronized))
                (x6 uhss (t-state on))
                (uhss hidro (v-uhss x7)(f-uhss y2)(fase-uhss y3)))))


(regra-16 1.0
        ((frame (tf-1  uhss (t-state on))
                (uhss hidro  (bus-B x1))
                (cb uhss  (cb1082 off)))
         (lisp (member 'x1 (list 'off 'on))))
        ((frame (cb uhss  (cb1082 on))
                (uhss  hidro (bus-B on)))))




(regra-17 1.0
        ((frame (tf-2  uhss (t-state on))
                (cb uhss  (cb1062 off))
                (uhss hidro  (bus-B x1)))
         (lisp (member 'x1 (list 'off 'on))))
        ((frame (cb uhss  (cb1062 on))
                (uhss  hidro (bus-B on)))))
```

```
(regra-18 1.0
        ((frame (tf-2  uhss (t-state on))
                (uhss hidro  (bus-B maintenance)
                                                (bus-A off))
                (cb uhss (cb1070 off))))
        ((frame (cb uhss (cb1070 on))
                (uhss hidro (bus-A on)))))

(regra-19 1.0
         ((frame (uhss  hidro (bus-B on))
                 (cb uhss  (cb1022 off))))
         ((frame (cb uhss  (cb1022 on))
                 (lt uhss  (lt-sgd on)))))

(regra-20 1.0
         ((frame (uhss  hidro (bus-B on))
                 (cb uhss  (cb1030 off))
                 (lt uhss  (lt-sgd on))
                 (mw uhss  (mw-sgd x1)))
          (lisp (> x1 0)))
         ((frame (cb uhss  (cb1030 on))
                 (lt uhss  (lt-ita on)))))


(regra-21 1.0
         ((frame (uhss  hidro (bus-A on)
                 (bus-B maintenance))
                 (cb uhss  (cb1032 off))))
         ((frame (cb uhss  (cb1032 on))
                 (lt uhss  (lt-ita on)))))


(regra-22 1.0
         ((frame (uhss  hidro (bus-A on)
                 (bus-B maintenance))
                 (cb uhss  (cb1030 off))
                 (lt uhss  (lt-ita on))
                 (mw uhss  (mw-ita x1)))
          (lisp (> x1 0)))
         ((frame (cb uhss  (cb1030 on))
                 (lt uhss  (lt-sgd on)))))
```

```
(regra-23 1.0
          ((logic (syncronize generator-group 1))
           (frame (uhss  hidro (x1 on))
                  (x2 x3 (state syncronized))
                  (x4 x3 (state off)))
           (lisp (and (member 'x1 (list 'bus-A 'bus-B)))))
          ((frame (x4 x3 (state on)))
           (logic (off (syncronize generator-group 1)))))

(regra-24 1.0
          ((logic (syncronize generator-group 1))
           (frame (uhss  hidro (x1 on))
                  (x2 x3 (state off))
                  (x4 x3 (state off)))
           (lisp (and (member 'x1 (list 'bus-A 'bus-B))
                      (string< (symbol-name 'x2) (symbol-name 'x4)))))
          ((frame (x2 x3 (state on)))
           (logic (off (syncronize generator-group 1)))))

(regra-25 1.0
          ((logic (syncronize generator-group 1))
           (frame (uhss  hidro (x1 on))
                  (x2 x3 (state on))
                  (x4 x3 (state off)))
           (lisp (and (member 'x1 (list 'bus-A 'bus-B))
                      (member 'x2 (list 'generator-1 'generator-2
                                        'generator-3 'generator-4))
                      (member 'x4 (list 'generator-1 'generator-2
                                        'generator-3 'generator-4)))))
          ((frame (x4 x3 (state on)))
           (logic (off (syncronize generator-group 1)))))

(regra-26 1.0
          ((logic ( recompouse lt uhss-ivp)
                  (total-syncronized-generator unknowed))
           (frame (cb uhss  (cb1090 off))
                  (cb uhss  (cb1092 off)))
           (lisp  (cond (t
                      (with-slots (message-to) (fase-msg *fase*)
                        (setf message-to 'ALL))
                    t))))
         ((export (request ((logic (external-syncronized-generator x y)))))
          (logic (off (total-syncronized-generator unknowed))
                 (off (update-generators off))
                 (update-generators on)
```

```
                (total-syncronized-generator required))))


(regra-27 1.0
          ((frame (network uhss (y1 died)))
           (logic (update-generators on)))
          ((logic (external-syncronized-generator  y1 0 ))))



(regra-28 1.0
         ((frame (generator-1 tf-1  (state x1))
                 (generator-2 tf-1  (state x2))
                 (generator-3 tf-2  (state x3))
                 (generator-4 tf-2  (state x4)))
          (logic (update-generators on))
          (include (?z . (n-times 'syncronized (list 'x1 'x2 'x3 'x4))))
          (lisp (eql 0 (n-times 'on (list 'x1 'x2 'x3 'x4 )))))
         ((logic (off ( internal-syncronized-generator uhss y)))
          (logic ( internal-syncronized-generator uhss ?z))))



(regra-29 1.0
          ((logic ( recompouse lt uhss-ivp))
           (logic ( internal-syncronized-generator uhss x1))
           (logic ( external-syncronized-generator sgd  x2))
           (logic ( external-syncronized-generator gbm  x3))
           (frame (uhss  hidro (bus-B on)))
           (lisp (cond ((< 3 (+ x1 x2 x3))
                        (with-slots (message-to) (fase-msg *fase*)
                        (setf message-to 'display))
                      t))))
          ((export (reply ((logic (recompoused lt uhss-ivp)))))
           (frame (cb uhss  (cb1090 on))
                  (lt uhss  (lt-ivp on)))
           (logic (off (recompouse lt uhss-ivp)))
           (logic (off ( internal-syncronized-generator uhss x1)))
           (logic (off ( external-syncronized-generator sdg x2)))
           (logic (off ( external-syncronized-generator gbm x3)))
           (logic (off (total-syncronized-generator required))
           (logic (total-syncronized-generator unknowed)))
           (logic  (off (update-generators on)))
           (logic  (update-generators off))))
```

```
(regra-30 1.0
          ((logic ( recompouse lt uhss-ivp))
           (logic ( internal-syncronized-generator uhss x1))
           (logic ( external-syncronized-generator sgd  x2))
           (logic ( external-syncronized-generator gbm  x3))
           (frame (uhss  hidro (bus-A on)))
           (lisp (cond ((< 3 (+ x1 x2 x3))
                  (with-slots (message-to) (fase-msg *fase*)
                        (setf message-to 'ivp))
                   t))))
          ((export (reply ((logic (recompoused lt uhss-ivp)))))
           (frame (cb uhss  (cb1092 on))
                  (lt uhss  (lt-ivp on)))
           (logic (off (recompouse lt uhss-ivp)))
           (logic (off (recompouse lt uhss-ivp)))
           (logic (off ( internal-syncronized-generator uhss x1)))
           (logic (off ( external-syncronized-generator sdg x2)))
           (logic (off ( external-syncronized-generator gbm x3)))
           (logic (off (total-syncronized-generator required))
           (logic  (off (update-generators on)))
           (logic  (update-generators off))
                  (total-syncronized-generator unknowed))))


(regra-31 1.0
          ((logic ( recompouse lt uhss-ivp))
           (logic ( internal-syncronized-generator uhss x1))
           (logic ( external-syncronized-generator sgd  x2))
           (logic ( external-syncronized-generator gbm  x3))
           (lisp (cond ((eql 3 (+ x1 x2 x3))
                  (with-slots (message-round message-to message-id)
                                    (fase-msg *fase*)
                        (setf message-round 0)
                        (setf message-id 0.2)
                        (setf message-to 'uhss))
                        t ))))
          ((export (request ((logic (syncronize-generator x 1)))))
           (logic  (off ( internal-syncronized-generator uhss x1)))
           (logic  (off ( external-syncronized-generator sgd  x2)))
           (logic  (off ( external-syncronized-generator gbm  x3)))
           (logic  (off (update-generators on)))
           (logic  (update-generators off))))
```

```
(regra-32 1.0
          ((logic ( recompouse lt uhss-ivp))
           (logic ( internal-syncronized-generator uhss x1))
           (logic ( external-syncronized-generator sgd  x2))
           (logic ( external-syncronized-generator gbm  x3))
           (frame (generator-1 tf-1  (state x4))
                  (generator-2 tf-1  (state x5))
                  (generator-3 tf-2  (state x6))
                  (generator-4 tf-2  (state x7)))
           (lisp (cond ((and (> 3 (+ x1 x2 x3))
                             (< (n-times 'on (list 'x4 'x5 'x6 'x7)) 4))
                        (with-slots (message-round message-to message-id)
                            (fase-msg *fase*)
                              (setf message-id 0.4)
                              (setf message-round 0)
                              (setf message-to 'uhss))))))
          ((export (request ((logic (syncronize-generator x 2)))))
           (logic  (off ( internal-syncronized-generator uhss x1)))
           (logic  (off ( external-syncronized-generator sgd  x2)))
           (logic  (off ( external-syncronized-generator gbm  x3)))
           (logic  (off (update-generators on)))
           (logic  (update-generators off))))
```

```
(regra-33 1.0
          ((logic (syncronize-generator x y))
           (frame (generator-1 tf-1  (state x1))
                  (generator-2 tf-1  (state x2))
                  (generator-3 tf-2  (state x3))
                  (generator-4 tf-2  (state x4)))
           (lisp (and  (eql 0 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (refuse ((logic (syncronize-generator uhss y)))))
           (logic (off (syncronize-generator x y)))))
```

```
(regra-34 0.8
          ((logic (syncronize-generator x y))
           (frame (generator-1 tf-1  (state x1))
                  (generator-2 tf-1  (state x2))
                  (generator-3 tf-2  (state x3))
                  (generator-4 tf-2  (state x4))
                  (uhss hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (< ( + y 1) (n-times 'off (list 'x1 'x2 'x3 'x4)))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator uhss y )))))
           (frame (uhss hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))

(regra-35 0.6
          ((logic (syncronize-generator x y))
           (frame (generator-1 tf-1  (state x1))
                  (generator-2 tf-1  (state x2))
                  (generator-3 tf-2  (state x3))
                  (generator-4 tf-2  (state x4))
                  (uhss hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (eql (+ 1 y) (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator uhss y)))))
           (frame (uhss hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))


(regra-36 0.4
          ((logic (syncronize-generator x y))
           (frame (generator-1 tf-1  (state x1))
                  (generator-2 tf-1  (state x2))
                  (generator-3 tf-2  (state x3))
                  (generator-4 tf-2  (state x4))
                  (uhss hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (eql y (n-times 'off (list 'x1 'x2 'x3 'x4)))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator uhss y)))))
           (frame (uhss hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))
```

```
(regra-37 1.0
          ((logic (external-syncronized-generator x y))
           (frame (generator-1 tf-1  (state x1))
                  (generator-2 tf-1  (state x2))
                  (generator-3 tf-2  (state x3))
                  (generator-4 tf-2  (state x4)))
           (include (?z . (n-times 'syncronized (list 'x1 'x2 'x3 'x4 ))))
           (lisp    (and (equal 'request (parla-primitive *fase*))
                         (not (equal (message-from (fase-msg *fase*))
                                     'uhss)))))
          ((export (reply ((logic (syncronized-generator uhss ?z)))))
           (logic (off (external-syncronized-generator x y)))))


(regra-38 1.0
          ((logic (syncronize-generator x y))
           (frame (uhss hidro (reserved-generators x6)))
           (include (?z . ( - x6 y )))
           (lisp (equal 'RECALL (parla-primitive *fase*))))
          ((frame (uhss hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))


(regra-39 1.0
          ((logic (syncronize-generator x y))
           (frame (uhss hidro (reserved-generators x6)))
           (include (?z . ( - x6 y )))
           (lisp (equal 'CONFIRM (parla-primitive *fase*))))
          ((frame (uhss hidro (reserved-generators ?z)))
           (logic (syncronize generator-group y ))
           (logic (off ((syncronize-generator x y))))))

(regra-40 1.0
          ((logic (syncronize-generator x1 2))
           (lisp (cond ((and (equal 'REPLY (parla-primitive *fase*))
                             (equal 'x1 'no-winner))
                        (with-slots (message-to message-round message-id)
                                    (fase-msg *fase*)
                          (setf message-id 0.1)
                          (setf message-to 'ALL)
                          (setf message-round 0))))))
           ((logic (off (syncronize-generator x1 2)))
            (export (request ((logic (syncronize-generator x 1)))))))
```

```
(regra-41 1.0
          ((logic (syncronize-generator x1 y1))
           (frame (network uhss (x2 alive)))
           (lisp (cond ((and (equal 'REPLY (parla-primitive *fase*))
                             (member 'x1 (list 'FASE 'SGD 'GBM)))
             (with-slots (message-to) (fase-msg *fase*)
                         (setf message-to 'ALL))))))
         ((export (request ((logic (external-syncronized-generator x y)))))
          (logic (off (syncronize-generator x1 y1)))
          (logic  (off (update-generators off)))
          (logic  (update-generators on))))

(regra-42 1.0
          ((logic (syncronize-generator x1 y1))
           (frame (network uhss (sgd died)
                                (gbm died)))
           (lisp (and (equal 'REPLY (parla-primitive *fase*))
                      (member 'x1 (list 'FASE 'SGD 'GBM)))))
          ((logic (off (syncronize-generator x1 y1)))
           (logic  (off (update-generators off)))
           (logic  (update-generators on))))
```

Apêndice B

Agente SGD

B.1 Construção Agente SGD

O processo de construção do *agente SGD* é similar ao do agente *UHSS*. As diferenças consistem apenas nos parâmetros passados, relacionados ao host onde o *agente* esta alocado.

B.2 Arquivo de fatos

```
((frame (hidro root)
        (sgd hidro
                          (v-nom 525)
                          (f-nom 60)
                          (v-sgd 0)
                          (f-sgd 0)
                          (fase-sgd 0)
                          (v-are 0)
                          (f-are 0)
                          (fase-are 0)
                          (v-uhss 0)
                          (f-uhss 0)
                          (fase-uhss 0)
             (site area-1)
             (bus-A off)
             (bus-B off)
             (reserved-generators 0))

             (generator-1 sgd
                          (tf  off)
                          (v 0)
                          (f 0)
                          (fase 0)
                          (state off))
```

```
        (generator-2 sgd
                     (tf  off)
                     (v 0)
                     (f 0)
                     (fase 0)
                     (state off))

        (generator-3 sgd
                     (tf  off)
                     (v 0)
                     (f 0)
                     (fase 0)
                     (state off))

        (generator-4 sgd
                     (tf off)
                     (v 0)
                     (f 0)
                     (fase 0)
                     (state off))


(LT sgd
             (lt-are off)
             (lt-uhss off))

(MW sgd
             (MW-are 0)
             (MW-uhss 0))

 (cb sgd
              (cb5205 off)
              (cb5208 off)
              (cb5219 off)
              (cb5222 off)
              (cb5225 off)
              (cb5233 off)
              (cb5236 off)
              (cb5239 off)
              (cb5247 off)
              (cb5250 off))))
```

B.3 Arquivo de regras

```
(regra-1 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-1 sgd  (state off))))
         ((frame (generator-1 sgd  (state on)))
          (logic (off (syncronize generator-group 1)))))


(regra-2 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-1 sgd  (state x1))
                 (generator-2 sgd  (state off)))
          (lisp (member 'x1 (list 'on 'syncronized 'maintenance))))
         ((frame (generator-2 sgd  (state on)))
          (logic (off (syncronize generator-group 1)))))


(regra-3 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-3 sgd (state off))
                 (generator-1 sgd  (state x1))
                 (generator-2 sgd  (state x2)))
          (lisp (and (member 'x1 (list 'on 'syncronized 'maintenance))
                     (member 'x2 (list 'on 'syncronized 'maintenance)))))
         ((frame (generator-3 sgd (state on)))
          (logic (off (syncronize generator-group 1)))))


(regra-4 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-4 sgd (state off))
                 (generator-1 sgd (state x1))
                 (generator-3 sgd  (state x2))
                 (generator-2 sgd  (state x3)))
          (lisp (and (member 'x1 (list 'on 'syncronized 'maintenance))
                     (member 'x2 (list 'on 'syncronized 'maintenance))
                     (member 'x3 (list 'on 'syncronized 'maintenance)))))
         ((frame (generator-4 sgd (state on)))
          (logic (off (syncronize generator-group 1)))))
```

```
(regra-5 1.0
        ((logic (syncronize generator-group y))
         (frame (generator-1 sgd  (state x1))
                (generator-2 sgd  (state x2))
                (generator-3 sgd  (state x3))
                (generator-4 sgd  (state x4))
                (y1 sgd  (state off)))
         (lisp (eql 'y (n-times 'off (list 'x1 'x2 'x3 'x4 )))))
        ((logic (off (syncronize generator-group 2)))
         (frame (y1 sgd  (state on)))))
```

```
(regra-6 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 sgd  (state x1))
                (generator-2 sgd  (state x2))
                (generator-3 sgd  (state x3))
                (generator-4 sgd  (state x4)))
         (lisp  (eql 4 (n-times 'off (list 'x1 'x2 'x3 'x4 )))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 sgd  (state on))
                (generator-2 sgd  (state on)))))
```

```
(regra-7 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 sgd  (state x1))
                (generator-2 sgd  (state x2))
                (generator-3 sgd  (state x3))
                (generator-4 sgd  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x1)
                    (equal 'off 'x2)
                    (equal 'off 'x3))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 sgd  (state on))
                (generator-2 sgd  (state on)))))
```

```
(regra-8 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 sgd  (state x1))
                (generator-2 sgd  (state x2))
                (generator-3 sgd  (state x3))
                (generator-4 sgd  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x1)
                    (equal 'off 'x2)
                    (equal 'off 'x4))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 sgd  (state on))
                (generator-2 sgd  (state on)))))


(regra-9 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 sgd  (state x1))
                (generator-2 sgd  (state x2))
                (generator-3 sgd  (state x3))
                (generator-4 sgd  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x1)
                    (equal 'off 'x3)
                    (equal 'off 'x4))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-3 sgd  (state on))
                (generator-4 sgd  (state on)))))


 (regra-10 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 sgd  (state x1))
                (generator-2 sgd  (state x2))
                (generator-3 sgd  (state x3))
                (generator-4 sgd  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x2)
                    (equal 'off 'x3)
                    (equal 'off 'x4))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-2 sgd  (state on))
                (generator-3 sgd  (state on)))))
```

```
(regra-11 1.0
        ((frame (sgd hidro (v-sgd x1)(f-nom x2)(v-nom x4))
                (x5 sgd (state on)(v x7)(f y1)(fase y2)))
         (lisp (and ( = 0 x1)
                    (> (* 0.1 x4) (abs (-  x7 x4)))
                    (> 0.2 (abs (- x2 y1))))))
        ((frame (x5 sgd (state syncronized)(tf on))
                (sgd hidro (v-sgd x7)(f-sgd y1)(fase-sgd y2)))))


(regra-12 1.0
        ((frame (sgd hidro (v-sgd x1)(f-sgd x2)(fase-sgd x4)(v-nom y1))
                (x5 sgd (state on)(v x7)(f y2)(fase y3)))
         (lisp (and (> x1 (* 0.9 y1))
                    (> (* 0.1 y1) (abs (-  x7 x1)))
                    (> 0.2 (abs (- x2 y2)))
                    (> 10 (abs (-  x4 y3))))))
        ((frame (x5 sgd (state syncronized)(tf on))
                (sgd hidro (v-sgd x7)(f-sgd y2)(fase-sgd y3)))))

(regra-13 1.0
        ((frame (sgd hidro (v-sgd x1)(f-nom x2)(v-nom x4))
                (sgd hidro  (v-uhss x7)(f-uhss y1)(fase-uhss y2))
                (sgd hidro  (bus-B off))
                (lt sgd (lt-uhss on)))
         (lisp (and ( = 0 x1)
                    (> (* 0.1 x4) (abs (-  x7 x4)))
                    (> 0.2 (abs (- x2 y1))))))
        ((frame (cb sgd (cb5239 on))
                (sgd hidro (v-sgd x7)(f-sgd y1)(fase-sgd y2)))))


(regra-14 1.0
        ((frame (sgd hidro (v-sgd x1)(f-sgd x2)(fase-sgd x4)(v-nom y1))
                (sgd hidro (v-uhss x7)(f-uhss y2)(fase-uhss y3))
                (sgd hidro  (bus-B on))
                (lt sgd (lt-uhss on))
                (cb sgd (cb5239 off)))
         (lisp (and (> x1 (* 0.9 y1))
                    (> (* 0.1 y1) (abs (-  x7 x1)))
                    (> 0.2 (abs (- x2 y2)))
                    (> 10 (abs (-  x4 y3))))))
        ((frame (cb sgd (cb5239 on))
                (sgd hidro (v-sgd x7)(f-sgd y2)(fase-sgd y3)))))
```

```
(regra-15 1.0
        ((frame (generator-1 sgd (tf on))
                (sgd hidro  (bus-B x1))
                (cb sgd  (cb5208 off)))
         (lisp (member 'x1 (list 'off 'on))))
        ((frame (cb sgd  (cb5208 on)))))

(regra-16 1.0
        ((frame (generator-1 sgd (tf on))
                (sgd hidro  (bus-A x1)(bus-B maintenance))
                (cb sgd  (cb5205 off)))
         (lisp (member 'x1 (list 'off 'on))))
        ((frame (cb sgd  (cb5205 on)))))

(regra-17 1.0
        ((frame (generator-4 sgd (tf on))
                (sgd hidro  (bus-B x1))
                (cb sgd  (cb5250 off)))
          (lisp (member 'x1 (list 'off 'on))))
         ((frame (cb sgd  (cb5250 on)))))

(regra-18 1.0
        ((frame (generator-4 sgd (tf on))
                (sgd hidro  (bus-A x1)(bus-B maintenance))
                (cb sgd  (cb5247 off)))
         (lisp (member 'x1 (list 'off 'on))))
        ((frame (cb sgd  (cb5247 on)))))


(regra-19 1.0
         ((frame (generator-2 sgd (tf on))
                 (sgd hidro  (bus-A x1)(bus-B maintenance))
                 (cb sgd  (cb5219 off)))
          (lisp (member 'x1 (list 'off 'on))))
         ((frame (cb sgd  (cb5219 on)(cb5222 on)))))

(regra-20 1.0
         ((frame (generator-2 sgd (tf on))
                 (sgd hidro  (bus-B x1))
                 (cb sgd  (cb5222 off)))
          (lisp (member 'x1 (list 'off 'on))))
         ((frame (cb sgd  (cb5222 on)))))
```

```
(regra-21 1.0
          ((frame (generator-3 sgd (tf on))
                  (sgd hidro  (bus-A x1)(bus-B maintenance))
                  (cb sgd  (cb5233 off)))
           (lisp (member 'x1 (list 'off 'on))))
          ((frame (cb sgd  (cb5233 on)))))
```

```
(regra-22 1.0
          ((frame (generator-3 sgd (tf on))
                  (sgd hidro  (bus-B x1))
                  (cb sgd  (cb5239 on))
                  (cb sgd  (cb5236 off)))
           (lisp (member 'x1 (list 'off 'on))))
          ((frame (cb sgd  (cb5236 on)))))
```

```
(regra-23 1.0
          ((frame (sgd  hidro (bus-B on))
                  (cb sgd  (cb5225 off))
                  (lt sgd  (lt-are off))))
          ((frame (cb sgd  (cb5225 on))
                  (lt sgd  (lt-are on)))))
```

```
(regra-24 1.0
          ((frame (sgd  hidro (bus-A on))
                  (cb sgd  (cb5219 on))
                  (cb sgd  (cb5222 on))
                  (lt sgd  (lt-are off))))
          ((frame (lt sgd  (lt-are on)))))
```

```
(regra-25 1.0
          ((frame (sgd  hidro (bus-B off))
                  (lt sgd  (lt-uhss x1))
                  (cb sgd  (cb5239 x2))
                  (generator-1 sgd (tf x3))
                  (cb sgd  (cb5208 x4))
                  (generator-2 sgd (tf x5))
                  (cb sgd  (cb5222 x6))
                  (cb sgd  (cb5225 x7))
                  (generator-3 sgd (tf y1))
                  (cb sgd  (cb5236 y2))
                  (generator-4 sgd (tf y3))
                  (cb sgd  (cb5250 y4)))
            (lisp (or (and (equal 'x1 'on) (equal 'x2 'on))
                      (and (equal 'x3 'on) (equal 'x4 'on))
                      (and (equal 'x5 'on) (equal 'x6 'on) (equal 'x7 'on))
                      (and (equal 'y1 'on) (equal 'y2 'on) (equal 'x2 'on))
                      (and (equal 'y3 'on) (equal 'y4 'on)))))
           ((frame (sgd  hidro (bus-B on))))))

(regra-26 1.0
          ((frame (sgd  hidro (bus-A off))
                  (lt sgd  (lt-uhss x1))
                  (cb sgd  (cb5236 x2))
                  (cb sgd  (cb5233 x3))
                  (generator-1 sgd (tf x4))
                  (cb sgd  (cb5205 x5))
                  (generator-2 sgd (tf x6))
                  (cb sgd  (cb5219 x7))
                  (generator-3 sgd (tf y1))
                  (generator-4 sgd (tf y2))
                  (cb sgd  (cb5247 y3)))
            (lisp (or (and (equal 'x1 'on) (equal 'x2 'on) (equal 'x3 'on))
                      (and (equal 'x4 'on) (equal 'x5 'on))
                      (and (equal 'x6 'on) (equal 'x7 'on))
                      (and (equal 'y1 'on) (equal 'x3 'on))
                      (and (equal 'y2 'on) (equal 'y3 'on)))))
           ((frame (sgd  hidro (bus-A on))))))

(regra-27 1.0
          ((frame (sgd  hidro (bus-A on))
                  (cb sgd  (cb5233 on))
                  (cb sgd  (cb5236 off))))
           ((frame (cb sgd  (cb5236 on))))))
```

```
(regra-28 1.0
          ((frame (sgd  hidro (bus-A off)(bus-B maintenance))
                  (lt sgd  (lt-uhss on))
                  (generator-3 sgd (state  off))))
          ((frame (generator-3 sgd (state on)))))



(regra-29 1.0
          ((frame (sgd  hidro (bus-A on))
                  (lt sgd  (lt-are off))
                  (generator-2 sgd (state  off))))
          ((frame (generator-2 sgd (state  on)))))



(regra-30 1.0
          ((logic (syncronize-generator x y))
           (frame (generator-1 sgd  (state x1))
                  (generator-2 sgd  (state x2))
                  (generator-3 sgd  (state x3))
                  (generator-4 sgd  (state x4)))
           (lisp  (and  (eql 0 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                        (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (refuse ((logic (syncronize-generator sgd y)))))
           (logic (off (syncronize-generator x y)))))



(regra-31 0.7
          ((logic (syncronize-generator x y))
           (frame (generator-1 sgd  (state x1))
                  (generator-2 sgd  (state x2))
                  (generator-3 sgd  (state x3))
                  (generator-4 sgd  (state x4))
                  (sgd hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (< ( + y 1) (n-times 'off (list 'x1 'x2 'x3 'x4)))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator sgd y )))))
           (frame (sgd hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))
```

```
(regra-32 0.5
          ((logic (syncronize-generator x y))
           (frame (generator-1 sgd  (state x1))
                  (generator-2 sgd  (state x2))
                  (generator-3 sgd  (state x3))
                  (generator-4 sgd  (state x4))
                  (sgd hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (eql (+ 1 y) (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator sgd y)))))
           (frame (sgd hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))


(regra-33 0.3
          ((logic (syncronize-generator x y))
           (frame (generator-1 sgd  (state x1))
                  (generator-2 sgd  (state x2))
                  (generator-3 sgd  (state x3))
                  (generator-4 sgd  (state x4))
                  (sgd hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (eql y (n-times 'off (list 'x1 'x2 'x3 'x4)))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator sgd y)))))
           (frame (sgd hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))


(regra-34 1.0
          ((logic (external-syncronized-generator x y))
           (frame (generator-1 sgd  (state x1))
                  (generator-2 sgd  (state x2))
                  (generator-3 sgd  (state x3))
                  (generator-4 sgd  (state x4)))
           (include (?z . (n-times 'syncronized (list 'x1 'x2 'x3 'x4 ))))
           (lisp  (and (equal 'request (parla-primitive *fase*))
                       (not (equal (message-from (fase-msg *fase*)) 'sgd))
                       (eql 0 (n-times 'on (list 'x1 'x2 'x3 'x4 ))))))
          ((export (reply ((logic (external-syncronized-generator sgd ?z)))))
           (logic (off (external-syncronized-generator x y)))))
```

```
(regra-35 1.0
          ((logic (syncronize-generator x y))
           (frame (sgd hidro (reserved-generators x6)))
           (include (?z . ( - x6 y )))
           (lisp (equal 'RECALL (parla-primitive *fase*))))
          ((frame (sgd hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))


(regra-36 1.0
          ((logic (syncronize-generator x y))
           (frame (sgd hidro (reserved-generators x6)))
           (include (?z . ( - x6 y )))
           (lisp (equal 'CONFIRM (parla-primitive *fase*))))
          ((frame (sgd hidro (reserved-generators ?z)))
           (logic (syncronize generator-group y ))
           (logic (off ((syncronize-generator x y))))))
```

Apêndice C

Agente GBM

C.1 Construção Agente GBM

O processo de construção do *agente GBM* é similar ao do agente *UHSS*. As diferenças consistem apenas nos parâmetros passados, relacionados ao host onde o *agente* esta alocado.

C.2 Arquivo de fatos

```
((frame (hidro root)
            (gbm hidro (v-nom 525)
                       (f-nom 60)
                       (v-gbm 0)
                       (f-gbm 0)
                       (fase-gbm 0)
                       (site area-1)
                       (bus-P off)
              (bus-PT off)
              (reserved-generators 0))

             (generator-1 gbm
                          (tf  off)
                          (v 0)
                          (f 0)
                          (fase 0)
                          (state off))

             (generator-2 gbm
                          (tf  off)
                          (v 0)
                          (f 0)
```

```
                (fase 0)
                (state off))

        (generator-3 gbm
                (tf  off)
                (v 0)
                (f 0)
                (fase 0)
                (state off))

        (generator-4 gbm
                (tf off)
                (v 0)
                (f 0)
                (fase 0)
                (state off))


(LT gbm
            (lt-are-1 off)
            (lt-are-2 off))

(MW gbm
            (MW-are-1 0)
            (MW-are-2 0))

 (cb gbm
             (cb5203 off)
             (cb5212 off)
             (cb5218 off)
             (cb5230 off)
             (cb5233 off)
             (cb5242 off)
             (cb5248 off))))
```

C.3 Arquivo de regras

```
(regra-1 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-1 gbm (state off))))
         ((frame (generator-1 gbm (state on)))
          (logic (off (syncronize generator-group 1)))))


(regra-2 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-1 gbm  (state x1))
                 (generator-2 gbm  (state off)))
          (lisp (member 'x1 (list 'on 'syncronized 'maintenence))))
         ((frame (generator-2 gbm  (state on)))
          (logic (off (syncronize generator-group 1)))))


(regra-3 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-3 gbm (state off))
                 (generator-1 gbm  (state x4))
                 (generator-2 gbm  (state x5)))
          (lisp (and (member 'x4 (list 'on 'syncronized 'maintenence))
                     (member 'x5 (list 'on 'syncronized 'maintenence)))))
         ((frame (generator-3 gbm (state on)))
          (logic (off (syncronize generator-group 1)))))


(regra-4 1.0
         ((logic (syncronize generator-group 1))
          (frame (generator-4 gbm (state off))
                 (generator-1 gbm (state x1))
                 (generator-3 gbm  (state x4))
                 (generator-2 gbm  (state x5)))
          (lisp (and (member 'x1 (list 'on 'syncronized 'maintenence))
                     (member 'x4 (list 'on 'syncronized 'maintenence))
                     (member 'x5 (list 'on 'syncronized 'maintenence)))))
         ((frame (generator-4 gbm (state on)))
          (logic (off (syncronize generator-group 1)))))
```

```
(regra-5 1.0
        ((logic (syncronize generator-group y))
         (frame (generator-1 gbm  (state x1))
                (generator-2 gbm  (state x2))
                (generator-3 gbm  (state x3))
                (generator-4 gbm  (state x4))
                (y1 gbm  (state off)))
         (lisp (eql 'y (n-times 'off (list 'x1 'x2 'x3 'x4 )))))
        ((logic (off (syncronize generator-group 2)))
         (frame (y1 gbm  (state on)))))


(regra-6 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 gbm  (state x1))
                (generator-2 gbm  (state x2))
                (generator-3 gbm  (state x3))
                (generator-4 gbm  (state x4)))
         (lisp (and (eql 4 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x1)
                    (equal 'off 'x2))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 gbm  (state on))
                (generator-2 gbm  (state on)))))


(regra-7 1.0
        ((logic (syncronize generator-group 2))
         (frame (generator-1 gbm  (state x1))
                (generator-2 gbm  (state x2))
                (generator-3 gbm  (state x3))
                (generator-4 gbm  (state x4)))
         (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                    (equal 'off 'x1)
                    (equal 'off 'x2)
                    (equal 'off 'x3))))
        ((logic (off (syncronize generator-group 2)))
         (frame (generator-1 gbm  (state on))
                (generator-2 gbm  (state on)))))
```

```
(regra-8 1.0
         ((logic (syncronize generator-group 2))
          (frame (generator-1 gbm  (state x1))
                 (generator-2 gbm  (state x2))
                 (generator-3 gbm  (state x3))
                 (generator-4 gbm  (state x4)))
          (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                     (equal 'off 'x2)
                     (equal 'off 'x3)
                     (equal 'off 'x4))))
         ((logic (off (syncronize generator-group 2)))
          (frame (generator-2 gbm  (state on))
                 (generator-3 gbm  (state on)))))



(regra-9 1.0
         ((logic (syncronize generator-group 2))
          (frame (generator-1 gbm  (state x1))
                 (generator-2 gbm  (state x2))
                 (generator-3 gbm  (state x3))
                 (generator-4 gbm  (state x4)))
          (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                     (equal 'off 'x1)
                     (equal 'off 'x2)
                     (equal 'off 'x4))))
         ((logic (off (syncronize generator-group 2)))
          (frame (generator-1 gbm  (state on))
                 (generator-2 gbm  (state on)))))



(regra-10 1.0
         ((logic (syncronize generator-group 2))
          (frame (generator-1 gbm  (state x1))
                 (generator-2 gbm  (state x2))
                 (generator-3 gbm  (state x3))
                 (generator-4 gbm  (state x4)))
          (lisp (and (eql 3 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                     (equal 'off 'x1)
                     (equal 'off 'x3)
                     (equal 'off 'x4))))
         ((logic (off (syncronize generator-group 2)))
          (frame (generator-1 gbm  (state on))
                 (generator-3 gbm  (state on)))))
```

```
(regra-11 1.0
        ((frame (gbm hidro (v-gbm x1)(f-nom x2)(v-nom x4))
                (x5 gbm (state on)(v x7)(f y1)(fase y2)))
         (lisp (and ( = 0 x1)
                    (> (* 0.1 x4) (abs (-  x7 x4)))
                    (> 0.2 (abs (- x2 y1))))))
        ((frame (x5 gbm (state syncronized)(tf on))
                (gbm hidro (v-gbm x7)(f-gbm y1)(fase-gbm y2)))))


(regra-12 1.0
        ((frame (gbm hidro (v-gbm x1)(f-gbm x2)(fase-gbm x4)(v-nom y1))
                (x5 gbm (state on)(v x7)(f y2)(fase y3)))
         (lisp (and (> x1 (* 0.9 y1))
                    (> (* 0.1 y1) (abs (-  x7 x1)))
                    (> 0.2 (abs (- x2 y2)))
                    (> 10 (abs (-  x4 y3))))))
        ((frame (x5 gbm (state syncronized)(tf on))
                (gbm hidro (v-gbm x7)(f-gbm y2)(fase-gbm y3)))))


(regra-13 1.0
        ((frame (generator-1 gbm (tf on))
                (cb gbm  (cb5203 off))))
        ((frame (cb gbm  (cb5203 on)))))


(regra-14 1.0
        ((frame (generator-2 gbm (tf on))
                (cb gbm  (cb5218 off))))
        ((frame (cb gbm  (cb5218 on)))))

(regra-15 1.0
        ((frame (generator-3 gbm (tf on))
                (cb gbm  (cb5233 off))))
        ((frame (cb gbm  (cb5233 on)))))



(regra-16 1.0
        ((frame (generator-4 gbm (tf on))
                (cb gbm  (cb5248 off))))
        ((frame (cb gbm  (cb5248 on)))))
```

```
(regra-17 1.0
          ((frame (gbm  hidro (bus-P off))
                  (cb gbm  (cb5203 x1))
                  (cb gbm  (cb5218 x2))
                  (cb gbm  (cb5233 x3))
                  (cb gbm  (cb5248 x4))
                  (cb gbm  (cb5230 off)))
           (lisp (or (equal 'x1 'on)
                     (equal 'x2 'on)
                     (equal 'x3 'on)
                     (equal 'x4 'on))))
          ((frame (gbm  hidro (bus-P on)))))


(regra-18 1.0
          ((frame (gbm  hidro (bus-PT off))
                  (cb gbm  (cb5203 x1))
                  (cb gbm  (cb5218 x2))
                  (cb gbm  (cb5233 x3))
                  (cb gbm  (cb5248 x4))
                  (cb gbm  (cb5230 on)))
           (lisp (or (equal 'x1 'on)
                     (equal 'x2 'on)
                     (equal 'x3 'on)
                     (equal 'x4 'on))))
          ((frame (gbm  hidro (bus-PT on)))))


(regra-19 1.0
          ((frame (gbm hidro (bus-PT x1)
                                              (bus-P x2))
                  (cb  gbm (cb5212 off)))
           (lisp (or (equal 'x1 'on) (equal 'x2 'on))))
          ((frame (cb gbm (cb5212 on)))))

(regra-20 1.0
          ((frame (cb gbm (cb5212 on))
                  (lt gbm (lt-are-1 off))))
          ((frame (lt gbm (lt-are-1 on)))))
```

```
(regra-21 1.0
          ((frame (gbm hidro (bus-PT x1)
                             (bus-P x2))
                  (cb gbm (cb5212 on)(cb5242 off))
                  (mw gbm (mw-are-1 x3))
                  (cb gbm  (cb5203 x5))
                  (cb gbm  (cb5218 x6))
                 (cb gbm  (cb5233 x7))
                  (cb gbm  (cb5248 y1)))
          (lisp (and (or (equal 'x1 'on) (equal 'x2 'on))
                     (> x3 0)
                     (or (and (equal 'x5 'on)
                              (equal 'x6 'on)
                              (equal 'x7 'on))
                         (and (equal 'x5 'on)
                              (equal 'x7 'on)
                              (equal 'y1 'on))
                         (and (equal 'x6 'on)
                              (equal 'x7 'on)
                              (equal 'y1 'on))
                         (and (equal 'x5 'on)
                              (equal 'x6 'on)
                              (equal 'y1 'on))))))
          ((frame (cb gbm (cb5242 on)))))


(regra-22 1.0
          ((frame (cb gbm (cb5242 on))
                  (lt gbm (lt-are-2 off))))
          ((frame (lt gbm (lt-are-2 on)))))



(regra-23 1.0
          ((logic (syncronize-generator x y))
           (frame (generator-1 gbm  (state x1))
                  (generator-2 gbm  (state x2))
                  (generator-3 gbm  (state x3))
                  (generator-4 gbm  (state x4)))
           (lisp (and
                   (eql 0 (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                   (equal 'ANNOUNCE (parla-primitive *fase*)))))
    ((export (refuse ((logic (syncronize-generator gbm y)))))
     (logic (off (syncronize-generator x y)))))
```

```
(regra-24 0.9
          ((logic (syncronize-generator x y))
           (frame (generator-1 gbm  (state x1))
                  (generator-2 gbm  (state x2))
                  (generator-3 gbm  (state x3))
                  (generator-4 gbm  (state x4))
                  (gbm hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (< ( + y 1)
                          (n-times 'off (list 'x1 'x2 'x3 'x4)))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
   ((export (acept ((logic (syncronize-generator gbm y )))))
    (frame (gbm hidro (reserved-generators ?z)))
    (logic (off (syncronize-generator x y)))))

(regra-25 0.7
          ((logic (syncronize-generator x y))
           (frame (generator-1 gbm  (state x1))
                  (generator-2 gbm  (state x2))
                  (generator-3 gbm  (state x3))
                  (generator-4 gbm  (state x4))
                  (gbm hidro (reserved-generators x5)))
           (include (?z . ( +  y x5)))
           (lisp (and  (eql (+ 1 y)
                            (n-times 'off (list 'x1 'x2 'x3 'x4 )))
                       (equal 'ANNOUNCE (parla-primitive *fase*)))))
          ((export (acept ((logic (syncronize-generator gbm y)))))
           (frame (gbm hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))

(regra-26 0.6
     ((logic (syncronize-generator x y))
      (frame (generator-1 gbm  (state x1))
                  (generator-2 gbm  (state x2))
                  (generator-3 gbm  (state x3))
                  (generator-4 gbm  (state x4))
                  (gbm hidro (reserved-generators x5)))
     (include (?z . ( +  y x5)))
     (lisp (and  (eql y (n-times 'off (list 'x1 'x2 'x3 'x4)))
                 (equal 'ANNOUNCE (parla-primitive *fase*))
            (not (equal (message-from (fase-msg *fase*)) 'gbm)))))
      ((export (acept ((logic (syncronize-generator gbm y)))))
       (frame (gbm hidro (reserved-generators ?z)))
       (logic (off (syncronize-generator x y))))))
```

```
(regra-27 1.0
          ((logic (external-syncronized-generator x y))
           (frame (generator-1 gbm  (state x1))
                  (generator-2 gbm  (state x2))
                  (generator-3 gbm  (state x3))
                  (generator-4 gbm  (state x4)))
           (include (?z . (n-times 'syncronized (list 'x1 'x2 'x3 'x4 ))))
           (lisp    (and (equal 'request (parla-primitive *fase*))
                         (not (equal (message-from (fase-msg *fase*)) 'gbm))
                         (eql 0 (n-times 'on (list 'x1 'x2 'x3 'x4 ))))))
          ((export (reply ((logic (external-syncronized-generator gbm ?z)))))
           (logic (off (external-syncronized-generator x y)))))

(regra-28 1.0
          ((logic (syncronize-generator x y))
           (frame (gbm hidro (reserved-generators x6)))
           (include (?z . ( - x6 y )))
           (lisp (equal 'RECALL (parla-primitive *fase*))))
          ((frame (gbm hidro (reserved-generators ?z)))
           (logic (off (syncronize-generator x y)))))


(regra-29 1.0
          ((logic (syncronize-generator x y))
           (frame (gbm hidro (reserved-generators x6)))
           (include (?z . ( - x6 y )))
           (lisp (equal 'CONFIRM (parla-primitive *fase*))))
          ((frame (gbm hidro (reserved-generators ?z)))
           (logic (off ((syncronize-generator x y))))
           (logic (syncronize generator-group y))))
```

Bibliografia

[AH88] G. Agha and C. Hewitt. Concurrent programming using actors: exploiting large-scale paralelism. In *Readings in distributed artificial intelligence*, 1988. San Mateo, Morgan Kaufmann Publishers Inc.

[BC97] G. Bittencourt and A. C. P. L. da Costa. Expert-coop: An environment for cognitive multi-agent systems. *Accepted in IFAC/IFIP MCPL'97, Conference on Management and Control of Production and Logistics*, October 1997.

[BF81] A. Barr and E.A. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume I-II. William Kaufmann Inc., Los Altos, California, 1981.

[BF94] M. Barbuceanu and M. S. Fox. Cool:a language for describimg coordination in multi agent. *EIL working paper*, pages 1–15, 1994.

[Bir93] K. Birman. The process group approach to realiable distributed computing. *Communications of the ACM*, 36(12):37–53, 1993.

[Bit94] G. Bittencourt. Logical inference through dual transformation. In *Workshop on Logic, Language, Information and Computation (WoLLIC'94)*, pages 7–9, 1994. Recife, PE, Brazil, July 28-30.

[Bit96] G. Bittencourt. *Inteligência Artificial - Ferramentas e Teorias.* Instituto de Computação, UNICAMP, Brasil, 1996.

[BM93] G. Bittencourt and M. Marengoni. A customizable tool for the generation of production-based systems. In G. Rzevski, J. Pastor, and R.A. Adey, editors, *Eighth International Conference on Applications of Artificial Intelligence in Engineering (AIENG'93), Vol. 1: Design, Methods and Techniques*, pages 337–352. Computational Mechanics Publications, Elsevier Applied Science, 1993. held in Toulouse, France, 29 June - 1 July, 1993.

[Boi92] O. Boissier. *Le problème du contrôle dans un système de vision integrè.* PhD thesis, Institut National Polytechnique de Grenoble, 1992.

[Bro86] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):435–453, March 1986.

[Car87] E. Cardozo. *DPSK: a kernel for distributed problem solving.* PhD thesis, CAED, Carniege Mellon University, 1987.

[CB97] A. C. P. L. da Costa and G. Bittencourt. Parla: A cooperation language for cognitive multi-agent systems. *Accepted in EPIA'97, 8th Portuguese Conference of Artificial Inteligence*, October 1997.

[CD90] J. A. Campbell and M. P. D'Inverno. Knowledge interchange protocol. In Y. Demazeuu and J. P. Muller, editors, *Decentralizad A. I.*, 1990. Amsterdam, Elsevier Science Publishers B. V. p. 63-80.

[CGR92] H. Coelho, G. Gaspar, and I. Ramos. Experiments on achieving communication in communities of autonomous agents. In *Proceedings of the IFIP WG 8.3 Working Confenrece on Decision Sipport Systems: Experiments and Expectations, Fontainbleau, France*, 1992.

[CL83] D. D. Corkill and V. R. Lesser. The distributed vehicle monitoring testbed: a tool for investigating distributed problem solving network. *AI Magazine*, 4(3):15–33, Fall 1983.

[CL87] P.R. Cohen and H.J. Levesque. Intention = choice + commitment. In *Proceedings of AAAI-87, Seattle*, pages 410–415, 1987.

[Dem93] Y. Demazeau. La plate-forme paco et ses applications. In Yves Demazeau and Anne Collinot, editors, *Actes de la 2ème Journée Nationale du PRC-GDR Intelligence Artificielle, Montpellier, France*, December 1993.

[DP80] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, 1980.

[DP88] D. Dubois and H. Prade. *Possibility Theory - An Approach to the Computerized Processing of Uncertainty.* Plenum Press, 1988.

[DR94] E.H. Durfee and J.S. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In *Proceedings of the International Workshop on Distributed Artificial Intelligence*, July 1994.

[Dur91] E.H. Durfee. The distributed artificial intelligence melting pot. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1301–1306, November 1991. Special Issue on Distributed Artificial Intelligence.

[Erm80] L. D. Erman. The hearsay-ii speech-understanding system: integrating knowledge to resolve uncertainty. *Computer Surveys*, 12(2):213–253, Jane 1980.

[FBL71] E.A. Feigenbaum, B.G. Buchanan, and J. Lederberg. On generality and problem solving : A case study using the dendral program. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 6, pages 165–190. Edimburgh University Press, Edimbourgh, GB, 1971.

[FG91] J. Ferber and L. Gasser. Intelligence artificielle distribuée, 1991. Tutorial Notes of the 11^th Conference on Expert Systems and their Applications, Avignon'91, France.

[FHRSW82] J. Fain, F. Hayes-Roth, H. Sowizral, and D.A. Waterman. Programming in rosie: An introduction by means of examples. Technical Report Report N-1646-ARPA, Rand Corporation, February 1982.

[FLM95] T. Finin, Y. Labrou, and J. Mayfield. *KQML as an Agent Communication Language.* MIT Press, Cambridge, 1995.

[FM77] C.L. Forgy and J. McDermott. Ops : A domain independent production system. In *Proceedings of IJCAI 5*, pages 933–939, 1977.

[Fox81] M. S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):70–80, January 1981.

[Gas87] L. Gasser. Mace: A flexible testbed for distributed ai research. In M. N. Hunhs, editor, *Distributed Artificial Intelligence.* Los Altos, Morgan Kaufmann Publishers Inc., 1987.

[Gen92] Gensym. *G2 Reference Manual.* GENSYM Coorporation, 125 Cambridge Park Drive, Cambridge, MA 02140, 1992.

[Gev87] W.B. Gevarter. The nature and evaluation of commercial expert systems building tools. *IEEE Computer*, pages 24–41, May, 1987.

[GI91] C. Gu and Toru. Ishida. An efficient procedure for theorem proving in propositional logic on vector computers. *Parallel Computing*, 17:983–5, 1991.

[Hew88] C. Hewitt. Offices are open systems. In A.H. Bond and L. Gasser, editors, *Readings in distributed artificial intelligence*, pages 321–329. San Mateo, Morgan Kaufmann Publishers Inc.,, 1988.

[HR85] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, July 1985.

[Ish92] T. Ishida. The tower of babel: towards organization-centered problem solving. In *International Workshop on Distribured Artificial Intelligence*, pages 141–153, 1992. Glen Arbor, Michigan.

[KIO96] K. Kuwabara, T. Ishida, and N. Osato. "agentalk: Describing multiagent coordination protocols.". In *Transactions of IEICE (The Institute of Electronics, Information and Communication Engineers)*, 1996. B-I.

[KN80] K. KONOLIGE and N.J. NILSSON. Multi-agent planning systems. In *NATIONAL CONFERENCE OF THE AMERICAN ASSOCIATION FOR ARTIFICIAL INTELIGENCE, 1, Stanfor, Proceedings. Menlo Park, AAAI p. 138-141*, March 1980.

[Lam78] L. Lamport. Time, clocks, and ordering of events. *Communications of ACM*, 21(7):558–565, July 1978.

[Len88] D.B. Lenat. Beings:knowledge as interacting experts. *BOND, A. H. and GASSER, L. ed. Reading in distributed artificial intelligence*, pages 161–168, 1988. Morgan Kaufmann Publishers Inc.

[Les94] V.R. Lesser. An overview of dai: Viewing distributed ai as distributed search. *Jornal of Japanese Society for Artificial Intelligence - Special Issue on Distributed Artificial Intelligence*, 5(4):392–400, 1994. R. Nakano and Doshita (eds.).

[Lev90] P. Levi. Architectures of individual and distributed autonomous agents. In T. Kanade, F.C.A Groen, and L.O. Hertzberger, editors, *Intelligent Autonomous Agents 2*, pages 315–324. IAS, Amsterdam, NL, 1990.

[MaC92] R.A. MaClachlan. *CMU Common Lisp User's Manual.* Carnegie Mellon University, Pittsburgh,PA, 1992.

[MH69] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In D. Michie and B. Meltzer, editors, *Machine Intelligence 4*, pages 463–502. Edimburgh University Press, Edimbourgh, GB, 1969.

[MIN89] N.H. MINSKY. The imposition of protocols over open distributed systems. *Internal Report, Rutgeers University*, pages 75–94, 1989.

[Mor73] T.P. Moran. The symbolic nature of visual imagery. In *Proceedings of IJCAI 3*, pages 472–477, 1973.

[New80] A. Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.

[OC91] E. Oliveira and R. Camacho. A shell for cooperating expert systems. *Expert Systems*, 8(2):75–85, May 1991.

[OPE95] GRUPO DE TRABALHO DE ANÁLISE E ELABORAÇÃO DE NORMAS DE OPERAÇÃO. InstruÇÃo de operaÇÃo, i.o. 023, recomposiÇÃo de Áreas de auto-restabelecimento da regioo sul. Technical report, Eletrosul, 1995.

[Pos43] E. Post. Formal reductions of the general combinatorial problem. *American Journal of Mathematics*, 65:197–268, 1943.

[SDB92] J.S. Sichman, Y. DEMAZEAU, and O. BOISSER. When can knowledge-based systems be called agents? *Anais do IX Seminario Brasileiro de Inteligencia Artificial*, pages 172–185, Outubro 1992. ISSN 0104-6500.

[Sho76] E.H. Shortliffe. *Computer-Based Medical Consultations : MYCIN.* American Elsevier, New York, 1976.

[Sho92] Y. Shoham. Agent oriented programming. *International Workshop on Distributed Artificial Inteligence*, pages 345–353, 1992. 11, Glenn Arbor, Michigan.

[Sic96] J.S. Sichman. A model for the decision phase of autonomous belief revesion in open multi-agent system. *Journal of Brazilian Computer Society*, 3(1):40–50, March 1996. ISSN 0104-6500.

[SJ84] G.L. Steele Jr. *Common LISP*. Digital Press, Burlington, 1984.

[Smi80] R.G. Smith. The contract net protocol:hifh-level communication and control in a distributed problem solving. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980.

[Ste92] M. H. Stefanini. Talismn: a mult-agent system governed by linguistic laws for natural language processing. In *International Conference on Computational Linguistic (COLIG)*, 1992. Nantes, France.

[Wat73] D.A. Waterman. Pas-ii reference manual. Technical report, Carnegie-Mellon University, June 1973. Computer Science Department Report.

[Wat86] D.A. Waterman. *A Guide to Expert Systems*. Addison-Wesley Publishing Company, Reading, MA, 1986.